ACIT5930

MASTER'S THESIS

in

Applied Computer and Information Technology (ACIT)

May 2023

Biomedical Engineering

Brain activation monitoring from dot-probe task using functional near-infrared spectroscopy

Sven Ivar Ougendal

Department of Computer Science

Faculty of Technology, Art and Design

1 PREFACE

This master's thesis was inspired by the growing importance of understanding mental health issues, particularly using technology to measure these disorders. Throughout this process, I have been fortunate to receive support and guidance from numerous individuals that have contributed significantly to the success of this project.

First and foremost, I would like to express my gratitude to my supervisors, Peyman Mirtaheri and Rune Jonassen. Their expertise and guidance have been invaluable in navigating this research topic.

I am also grateful to Ph.D. student Elise Solbu Roalsø for her contributions to the recruitment process and experiment logistics, Morten Ødegård for his assistance in acquiring the necessary equipment, and Haroon Khan for always being available to answer questions.

Lastly, I would like to give special thanks to Ph.D. student Sandra Klonteig for being an amazing teammate throughout the entire process.

Place and date: 14.05.2023

Signature:

Sven Ivar
Ougendal

2 ABSTRACT

Over the past decade, there has been a notable increase of 13% in mental health issues. Approximately one in five children and adolescents globally suffer from a mental health problem, with suicide ranking as the second most common cause of death for individuals aged 15-29.

Cognitive biases, such as attentional bias (AB), may contribute to the onset and persistence of mental health disorders. AB can be defined as the tendency to selectively attend to or focus on certain stimuli while ignoring others, where someone with a negative AB have a disproportional attention to negative stimuli. The dot-probe task is one of the most widely used tasks to measure AB. Understanding the underlying neural processes involved in AB and identifying reliable biomarkers may be important in developing successful interventions for mental health disorders.

This master's thesis aimed to setup an experiment laboratory to investigate AB and its underlying neural mechanisms using a combined approach of hybrid functional near-infrared spectroscopy (fNIRS)/eye-tracking (ET) and hybrid electroencephalography (EEG)/ET systems in conjunction with the dot-probe task.

A large-scale experiment was conducted, and a multi-subject analysis was done on the fNIRS data, which focused on detecting significant variations in oxyhemoglobin (HbO) and deoxyhemoglobin (HbR) concentrations between congruent and incongruent dot-probe trials in the prefrontal cortex (PFC) and visual cortex regions of the brain.

The results of the multi-subject general linear model (GLM) analysis revealed two significant findings: (1) a higher HbR concentration in the right ventral medial PFC when the dot is located behind the fearful face compared to when it is behind the neutral face, and (2) a higher HbR concentration in the right dorsal PFC when the dot is positioned behind a face expressing emotion compared to when it is behind a neutral face. These findings highlight the potential of using fNIRS to study AB.

3 TABLE OF CONTENTS

1 Preface 2

2 Abstract 3

4 List of figures 7

5 List of tables 9

6 Introduction 10

7 State of the art 15

7.1 Attentional Bias (AB) 15

7.2 Previous Research on methods to Measure AB 16

7.3 Optical approach using near-infrared light 21

8 Methods 26

8.1 Setting up the lab 26

8.2 Experiment paradigm design 29

8.2.1 Time and signal synchronization 35

8.3 Experiment Protocol 38

8.3.1 Step-by-step guide 38

8.3.2 Experiment Order 38

8.3.3 Procedure for each modality 39

8.4 Test-run and Improvements made 44

8.4.1 Hybrid lab setup 44

8.4.2 Decreasing fNIRS sampling rate 45

8.4.3 Changing fNIRS paradigm to block design 46

8.4.4 Reducing fNIRS trigger setup 46

8.4.5 Integrating ET data with analysis software 47

8.4.6 Figure size and location 48

8.4.7 Making the experiment more engaging 49
8.4.8 Stimuli duration 50
8.4.9 Tiredness VAS-scale 50
8.4.10 Response-box 51
8.5 Conducting experiment 51
8.5.1 Ethics and storing of data 52
8.5.2 Participants and recruitment process 52
8.5.3 Procedure 53
8.6 Pre-processing and Analysis of fNIRS data 53
9 Results and Discussion 56
9.1 The finished lab setup and protocol 56
9.2 The finished experiment paradigm design 59
9.3 fNIRS pre-processing pipeline 63
9.3.1 Remove masked channels 64
9.3.2 Trimming of data 65
9.3.3 Channel rejection 65
9.3.4 Conversion of data 67
9.3.5 Edit events 67
9.3.6 Motion Correction 69
9.3.7 Physiological Noise removal 71
9.3.8 Normalization 73
9.3.9 Finished pre-processed data 74
9.4 Results from analysis 75
10 Conclusion 80
11 Future work 81
12 References 82

13 Appendix 94

A.1 Step-by-Step guide for experiment procedure 94

A.2 Research Paper.. 116

A.3 fNIRS/ET psychopy code.. 129

A.4 EEG/ET psychopy code.. 210

4 LIST OF FIGURES

Figure 6-1 Illustration of neurovascular coupling, from (Khan, Naseer, et al., 2021). 11

Figure 6-2 Illustration of the process to get results from cognitive neuroscience experiment. 13

Figure 7-1 Illustration of the different ERP components, from (Black, 2022). 19

Figure 7-2 Example of emitter-detector pairs showing the “banana-shaped” paths of light, image taken from (Naseer & Hong, 2015). 22

Figure 7-3 Ilustration of the absorption spectra of HbO (red) and HbR (blue) 23

Figure 7-4 Illustration the three types fNIRS techniques, picture taken from (Scholkmann, 2012). 24

Figure 8-1 A visualization of our montage configuration, with LEDs as red and photodiodes as green, as depicted in the NIRStar 15-3 software interface. 27

Figure 8-2 Illustration of Routines and Loops in a Dot-Probe Task.. 30

Figure 8-3 Overview of the dot-probe task, picture taken from (Andrzejewski & Carlson, 2020) with some edits to fit our setup. 31

Figure 8-4 An Excel file containing PsychoPy variables, as sourced from an Excel spreadsheet. 33

Figure 8-5 An Excel list produced by PsychoPy, as extracted from an Excel spreadsheet. 35

Figure 8-6 Exemplifies a trigger event (stippled line) in NirStar 15-3, as presented in the NIRx Trigger Manual (NIRx, 2019b). 36

Figure 8-7 Displays the head coordinate system utilized by both EEG and fNIRS setups, as depicted in the NIRx NIRSCap User Guide (NIRx, 2019). 39

Figure 8-8 (left) Demonstrates the technique for utilizing a q-tip to clear away hair, as presented in the NIRx Troubleshooting Signal Quality Getting Started Guide (NIRx, 2017). 40

Figure 8-9 (right) Depicts a participant engaged in the dot-probe task, wearing the fNIRS cap with shower-cap, captured during the test-run. 40

Figure 8-10 (left) Illustrates the process of applying EEG gel to the electrode, captured during the test-run... 42

Figure 8-11 (right) Displays the screen view while conducting an impedance measurement, as seen in g.recorder. 42

Figure 8-12 This illustration demonstrates the optimal appearance when the participant gazes at the corner of the screen... 43

Figure 8-13 Depicts the illumination patterns at different sampling rates. 45

Figure 8-14 Visual representation of the angular measurements displayed on the screen, adapted from (SR-Research, 2023)... 48

Figure 8-15 This image showcases the response box, as utilized in our study. The photograph has been captured from the MedTec test Lab. 51

Figure 9-1 Overview of the test lab.. 57

Figure 9-2 Showcase of the test lab, illustrating the participant's seating arrangement during testing.. 58

Figure 9-3 Showcase a person during the pre-calibration of fNIRS......................... 59

Figure 9-4 Illustration of the hybrid fNIRS/ET paradigm design............................... 61

Figure 9-5 Illustration of the hybrid EEG/ET paradigm design................................. 61

Figure 9-6 Illustration of workflow 1 64

Figure 9-7 Illustration of workflow 2 64

Figure 9-8 Illustration of data processing in Workflow 1 68

Figure 9-9 Comparative visualization of TDDR and spike-correction methods,........ 71

Figure 9-10 The comparative visualization showcases the impact of applying Workflow 2 to preprocess the data. 74

Figure 9-11 3D illustration of significant differences in contras map 2...................... 77

Figure 9-12 2D illustration of significant differences in contras map 2...................... 77

Figure 9-13 3D illustration of significant differences in contras map 4. 78

Figure 9-14 2D illustration of significant differences in contras map 4...................... 79

5 LIST OF TABLES

Table 8-1 An overview of the relationship between trigger signals and stimuli......... 38

Table 9-1 Overview of the new and improved fNIRS trigger setup. 62

Table 9-2 Overview of the EEG and ET trigger setup... 63

6 INTRODUCTION

Over the past decade, there has been a notable increase of 13% in mental health issues. Approximately one in five children and adolescents globally suffer from a mental health problem, with suicide ranking as the second most common cause of death for individuals aged 15-29. Depression and anxiety have a combined economic impact of US$ 1 trillion annually (WHO, 2023).

Cognitive biases, such as AB, may contribute to the onset and persistence of mental health disorders. AB can be defined as the tendency to selectively attend to or focus on certain stimuli while ignoring others, where someone with a negative AB have a disproportional attention to negative stimuli. For instance, individuals with depression often focus disproportionately on dysphoric stimuli, while those with anxiety are prone to being easily side-tracked by potential threats (Barry et al., 2015; Disner et al., 2011).

Understanding the underlying neural processes involved in AB and identifying reliable biomarkers may be important in developing successful interventions for mental health disorders. By incorporating cost-efficient and portable cognitive and neurofunctional measures of AB alongside conventional mental health evaluations, we can potentially enhance the accuracy of individualized treatment response predictions (Barry et al., 2015).

Numerous computerized experimental tasks have been utilized by researchers to investigate AB. The tasks employed include spatial cueing-, visual search-, Stroop-, and dot-probe tasks (Chew, 2015). Among these, the dot-probe task is regarded as the "gold standard" by some experts, as it remains the most widely employed technique in AB research (Kappenman et al., 2014; Torrence & Troup, 2018). The task employs reaction time (RT) as an indicator of AB. Nevertheless, relying solely on behavioural measures, such as RT, presents certain limitations. Incorporating supplementary assessment methods could offer additional insights into AB (Carlson, 2021; Carlson & Fang, 2020).

To better understand AB, measurement of different brain activities has been used in combination with the dot-probe task, mainly electrical and perfusion. Electrical activity refers to the changes in electrical potentials generated by neurons, which can be measured using EEG. Perfusion changes refer to the changes in blood flow and oxygenation levels in the brain, which can be measured using functional magnetic resonance imaging (fMRI) and fNIRS (Carlson & Fang, 2020; Price et al., 2014; Torrence, 2015). Electrical and perfusion changes are linked together through neurovascular coupling, where an increase in neuronal activity drives changes in blood flow and oxygenation to meet the demands of active brain regions, resulting in concentration changes in HbO and HbR (Khan, Naseer, et al., 2021).

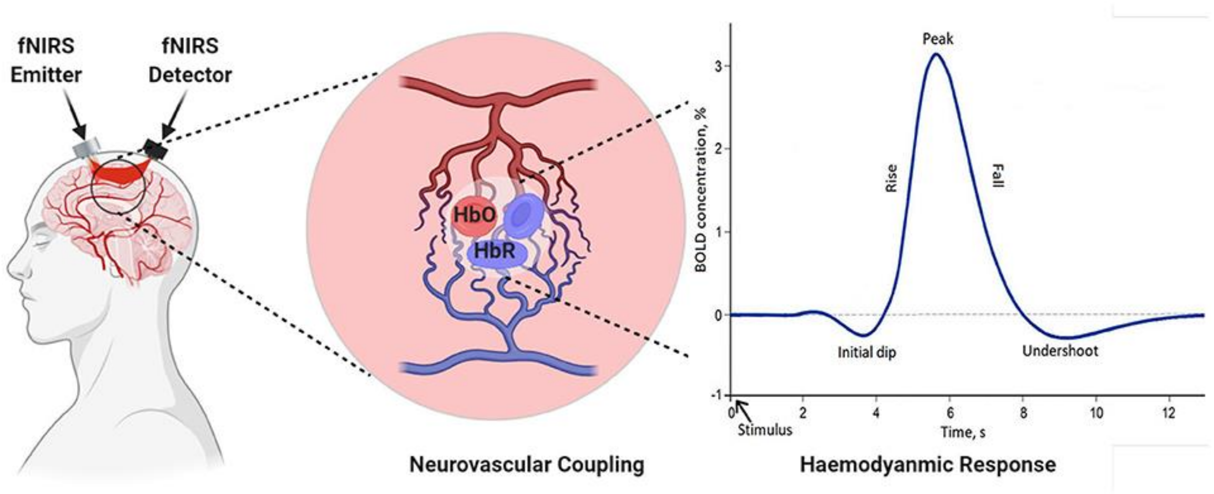


Figure 6-1 Illustration of neurovascular coupling, from (Khan, Naseer, et al., 2021).

Although fNIRS is relatively new and unexplored in AB research, it presents several advantages over fMRI, including measurements of both HbO and HbR, superior temporal resolution, substantially cheaper equipment, and the capacity to assess changes in cortical regions   in more natural settings compared to fMRI machines (Ehlis et al., 2014). fNIRS uses atleast two wavelength of light to measure concentration changes in HbO and HbR. The procedure involves sending near-infrared light into the brain tissue through the skull, where Hb absorbs it in the blood. Depending on the oxygenation status of the Hb, different amounts of light are absorbed by it. fNIRS can determine which regions of the brain are active during a specific task or at rest by observing these changes (Quaresima & Ferrari, 2019).

ET is also a measurement technique used in combination with the dot-probe task. The ET uses light, often infrared or near-infrared, to shine on the eye. It then detects the reflections from the cornea and pupil with sensors or cameras, tracking their position and direction (Carter & Luke, 2020). It measures eye movements and fixation patterns, providing information about where the participant is attending (Duque & Vázquez, 2015).

A hybrid EEG/fNIRS approach can be beneficial, as it allows for the simultaneous measurement of electrical and perfusion-based brain changes. However, this approach is in the early stages, where the hardware design needs improvements (Liu et al., 2021). It has proven challenging to record neural activity from the exact location (Khan, Naseer, et al., 2021). A viable alternative might be integrating hybrid fNIRS/ET and hybrid EEG/ET systems, as ET does not interfere with the signal quality or setup of the other modalities. This combined approach could enhance our comprehension of AB and their associated neural mechanisms by capitalizing on the strengths of each technique without incurring any drawbacks from individual modalities. Nevertheless, a recent systematic review on attention bias modification has endorsed the adoption of a multimodal approach for evaluating AB (Carlson, 2021).

A cognitive neuroscience experiment requires extensive preparation and multiple steps before a research question can be answered. The process begins with establishing a test laboratory, where all the necessary hardware and software must be installed and ready for use. This state-of-the-art environment is essential for producing accurate results, and ensuring the experiment runs in controlled surroundings.

Following the laboratory setup, an experiment paradigm must be designed to effectively display the stimulus to the participant. This is a crucial aspect of the process, as it determines the type of responses elicited from the subjects. Subsequently, a well-structured lab protocol must be developed, which provides an accessible overview of the experiment's procedure, enabling seamless data collection.

Once these preparatory steps are completed, researchers can begin collecting participant data. This information then undergoes a pre-processing stage, cleaned and organized to facilitate accurate analysis. The data is subsequently analysed, allowing researchers to identify patterns, correlations, and other significant findings. Finally, based on the analysis results, a conclusion can be drawn that contributes to our understanding of cognitive neuroscience and potentially informs future research in the field.

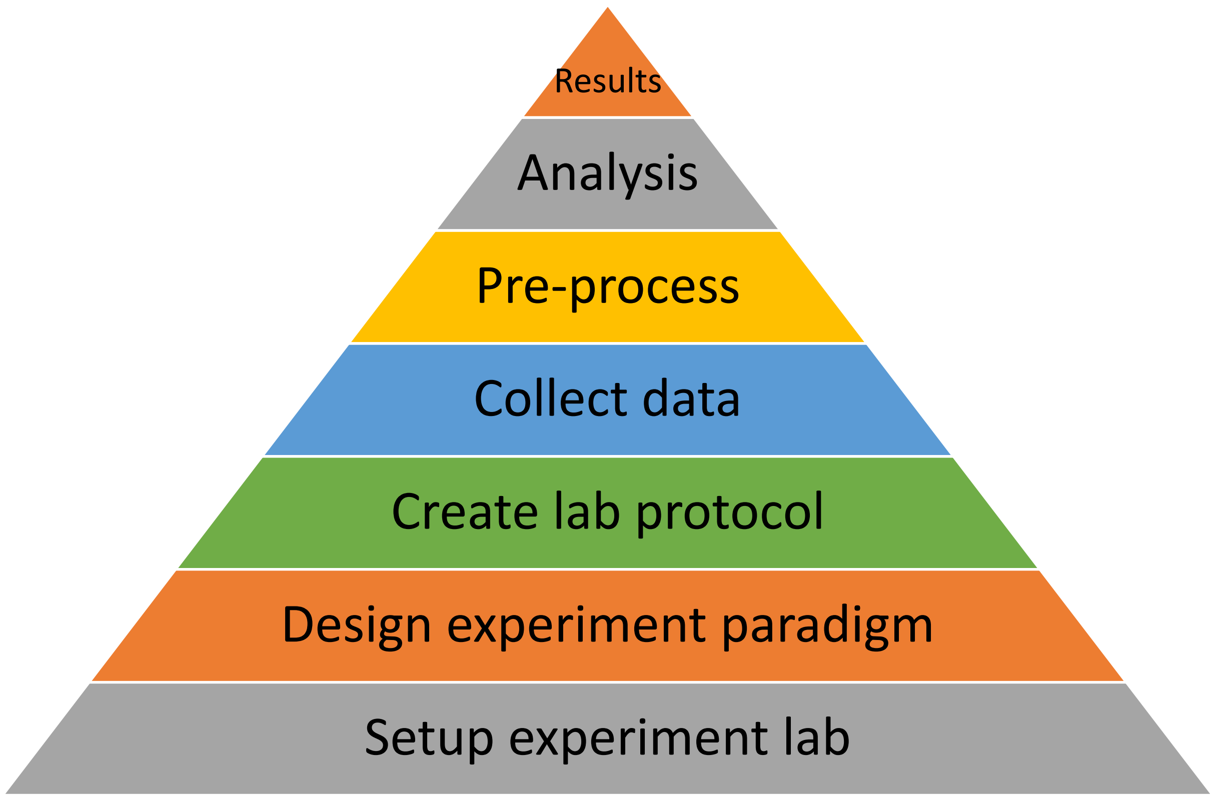


Figure 6-2 Illustration of the process to get results from cognitive neuroscience experiment.

The objectives of this master's thesis are to set up an environment to investigate and enhance our understanding of AB and their underlying neural mechanisms using a combined approach of hybrid fNIRS/ET and hybrid EEG/ET systems in conjunction with the dot-probe task. To accomplish this, the study aims to establish a state-of-the-art test laboratory, design an effective dot-probe experiment paradigm and create an experiment procedure pipeline that facilitates seamless data collection, pre-processing. By conducting a large-scale experiment with a considerable amount of data collected, this research aims to contribute to cognitive neuroscience and improve the clinical potential of AB measures.

To summarize, the key objectives of this master's thesis are as follows:

1. Establish a test laboratory with hybrid fNIRS/ET and EEG/ET systems.
2. Design a dot-probe experiment paradigm and create a comprehensive experiment procedure pipeline.
3. Conduct a large-scale experiment using the combined approach of hybrid fNIRS/ET and hybrid EEG/ET systems with the dot-probe task.
4. Develop an easy-to-use pipeline for fNIRS data pre-processing.
5. Perform a multi-subject analysis on the collected fNIRS data, where the research question is:
   Is it possible to detect any significant variations in HbO or HbR between congruent and incongruent dot-probe trials in the PFC and visual cortex regions of the brain?

ADvanced hEalth intelligence and brain-insPired Technologies (ADEPT), is a research group, situated at Oslomet, focused on developing and applying brain-inspired technologies to improve health outcomes. The research focus of ADEPT falls under two main categories: Brain Health and Brain-Inspired Technologies.

A robust and easy-to-use pipeline for data collection and pre-processing will contribute to ADEPT's research ambitions. By streamlining the data acquisition process and ensuring high-quality pre-processed data, researchers can focus on developing advanced brain-inspired technologies and applications more efficiently. This, in turn, will accelerate the rate at which these new technologies are integrated into the healthcare industry, ultimately leading to improved health outcomes for patients.

7 STATE OF THE ART

7.1 ATTENTIONAL BIAS (AB)

The cognitive theory suggests that anxiety, depression, and other mental illnesses are linked to AB. This means that people with these mental health issues tend to focus more on certain negative things while ignoring others, which can be influenced by a combination of genetic and environmental factors. The cognitive theory of anxiety and depression posits that AB are significant factors in the development and maintenance of these mental health conditions (Disner et al., 2011).

Attentional selection falls into two types: top-down and bottom-up. Top-down attentional selection is voluntary and goal-oriented, while bottom-up attentional selection is automatic and driven by attentional capture (Weierich et al., 2008). People with anxiety often show a stronger inclination towards bottom-up processing, especially when confronted with potential threats (Eysenck et al., 2007). These biases can appear as difficulties in inhibiting distractions caused by emotional stimuli, increased shifting of attention towards or away from threatening stimuli, or problems in updating attention based on new data (Cisler & Koster, 2010).

Anxiety research often highlights the prioritization of threat-related stimuli, whereas depression research emphasizes the prioritization of negative information and diminished responsiveness to positive stimuli (T. Armstrong & B. O. Olatunji, 2012; Peckham et al., 2010). However, the role of AB in depression remains inconsistent, with some studies examining the potential interaction effects of co-existing anxiety in depression-related AB (Sass et al., 2014).

Transdiagnostic is an approach to mental health research that suggests common cognitive, behavioural, and psychophysiological processes underlie symptoms across different diagnoses (Garland & Howard, 2014; Mansell et al., 2008). AB exemplify a cognitive transdiagnostic process, where a systematic review found that research on AB has been conducted in relation to anxiety, depression, eating disorders, obsessive-compulsive disorder, and addictive disorders (Rogers et al., 2020). This perspective emphasizes studying transdiagnostic processes, such as AB, to inform future transdiagnostic treatments (Craske, 2012).

7.2 Previous Research on methods to Measure AB

Numerous methodologies have been employed by researchers to examine AB. The techniques implemented encompass spatial cueing, visual searching, Stroop, and dot-probe task. The latter two, namely the Stroop and dot-probe tasks, have gained the most widespread recognition and usage (Chew, 2015).

The Stroop task assesses AB by evaluating the RT at which participants identify the colours of displayed words. In an adapted version of this task, researchers employ disorder-specific words to investigate AB in psychological disorders such as anxiety and depression. Findings indicate that individuals with these conditions demonstrate slower colour-naming responses for disorder-relevant words in comparison to neutral words (Gotlib & McCann, 1984; Mattia et al., 1993). This delay is thought to reflect an AB toward disorder-relevant words (Williams et al., 1996).

However, there are some limitations to the Stroop task. One limitation is that it does not provide information about the specific attentional processes involved or differentiate between different attentional processes (De Ruiter & Brosschot, 1994; Dobson & Dozois, 2004). Another limitation is the ecological validity of using word stimuli, which may not accurately represent real-life situations (Thomas Armstrong & Bunmi O. Olatunji, 2012). Additionally, the Stroop task provides only a single snapshot of the attentional process, which may not fully capture the complexity of the AB (Thomas Armstrong & Bunmi O. Olatunji, 2012).

The dot-probe task involves presenting two stimuli (pictures, words, etc.) simultaneously for a moment, followed by the appearance of a small dot over one of the stimuli. The participant is instructed to respond as quickly as possible by pushing a button when the dot appears. The difference in RT between dots appearing after emotional- vs. neutral- stimuli provides an index of AB, with shorter RTs indicating bias toward the cue (Cisler & Koster, 2010).

The primary benefit of the dot probe task over the Stroop task lies in its flexibility. The Stroop task relies on colour-naming responses, which restricts its stimuli to words only (Chew, 2015). This constraint may not adequately capture the spectrum of anxiety-inducing stimuli for those experiencing anxiety (Bradley et al., 2000). In contrast, the dot probe task offers a more targeted assessment of AB, as it enables the distinction between attentional vigilance and avoidance (Jiang & Vartanian, 2018).

However, traditional methods for assessing AB using RT indices have demonstrated weak internal consistency and limited test-retest dependability, as reported in various studies (Brown et al., 2014; Schmukle, 2005; Staugaard, 2009). Researchers have also investigated other RT-based metrics, such as those based on variability, but these alternatives have proven to be inconsistent as well (Carlson & Fang, 2020; Naim et al., 2015; Price et al., 2015).

Given the inadequacies of existing methods, other measuring methods of AB has been tested. Researchers have explored other techniques, including subjective evaluations of stimuli, eye-movement biases, event-related potentials (ERP), and neuroimaging approaches such as fMRI to study neural activation patterns and connectivity (Thomas Armstrong & Bunmi O. Olatunji, 2012; Britton et al., 2013; Carlson & Fang, 2020; Price et al., 2014; Torrence & Troup, 2018). Technologies like ET, EEG, and fNIRS have been proposed as potential supplements to improve the dot-probe task measurements of AB (Carlson, 2021).

ET offer a direct way to observe participants' eye movement patterns, allowing for a relatively immediate and ongoing evaluation of explicit visual focus (Thomas Armstrong & Bunmi O. Olatunji, 2012). Common indices for ET include the duration of gaze (dwell time) and the quantity or latency of the initial fixation. In the context of anxiety, the vigilance hypothesis posits that attention is initially directed towards threatening stimuli, while the maintenance hypothesis proposes that cognitive resources remain focused on such stimuli (Thomas Armstrong & Bunmi O. Olatunji, 2012; Fox et al., 2001). However, these hypotheses may not hold as much significance for individuals with depression.

For depressive individuals, stimulus relevance has not been shown to cause an immediate allocation of attention. Instead, they often exhibit diminished orientation and briefer gaze maintenance on positively charged stimuli (e.g., happy faces), while displaying increased gaze maintenance on negatively charged stimuli (e.g., sad faces) (Thomas Armstrong & Bunmi O. Olatunji, 2012; Duque & Vázquez, 2015). One potential drawback of ET methods is their dependence on overt behavior, which may not accurately reflect covert attention processes (Thomas Armstrong & Bunmi

O. Olatunji, 2012). Nevertheless, studies examining ET reliability as an AB measurement tool have yielded encouraging outcomes, with some research suggesting it possesses greater validity and dependability compared to reaction time (Price et al., 2015; Waechter et al., 2014).

EEG is a non-invasive method of measuring the neural activity of the brain using scalp electrodes (Luck, 2014). The electrical activity captured by EEG can be divided into event-related potentials (ERPs), which are time-locked and averaged around an event or stimulus. ERPs offer benefits such as high temporal resolution and the ability to reflect the degree of processing through amplitude difference measurements. Neural chronometry of AB implies that distinct ERP components correspond to separate stages of information processing. Early stages of sensory processing are associated with the P1, N1, N170 (N1 component linked to processing of faces), and N2pc (N2 component linked to selective attention) components, typically seen in posterior or sensory regions. In contrast, later stages of strategic processing, such as engagement and disengagement processes, are linked to P2, N2, and P3 components, usually detected in anterior or frontal areas (Carlson, 2021; Gupta et al., 2019; Torrence & Troup, 2018).

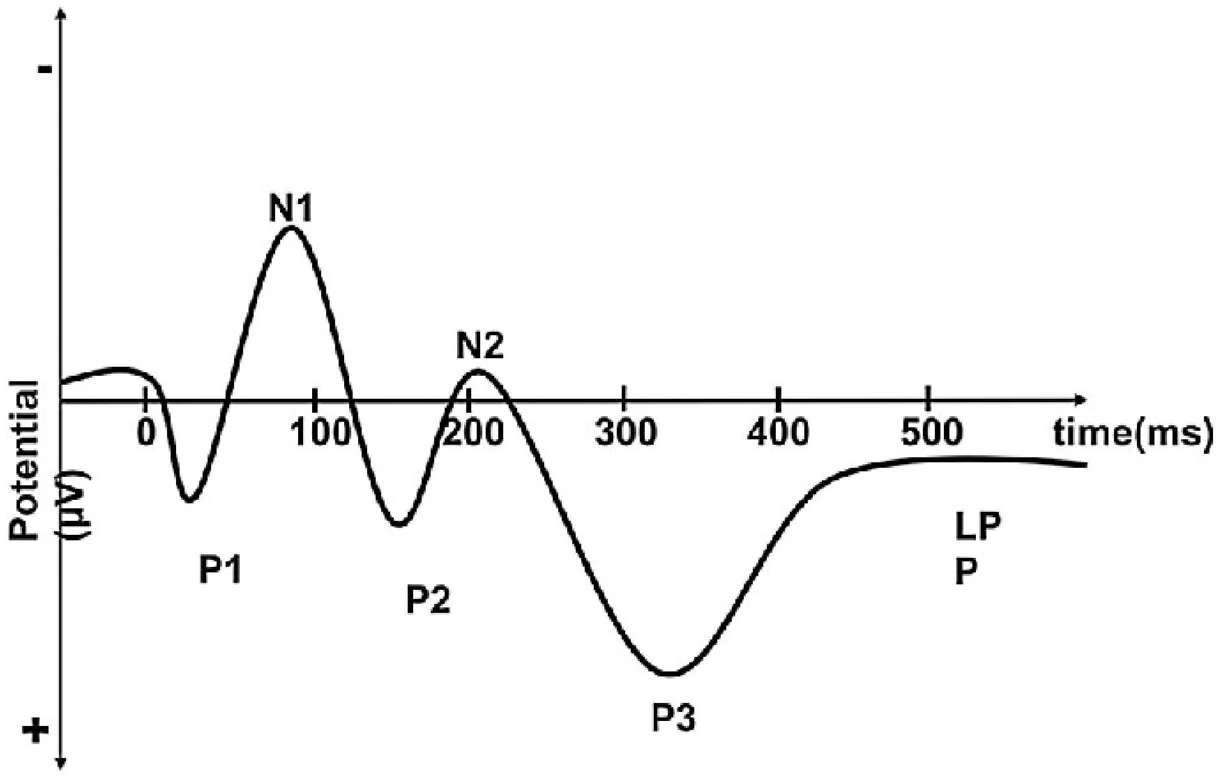


Figure 7-1 Illustration of the different ERP components, from (Black, 2022).

Recent literature reviews indicate a growing interest in using ERPs as an AB outcome measure, with some components showing potential for valid and reliable measurements (Carlson, 2021; Torrence & Troup, 2018). For instance, the N2pc component has been recognized as a more dependable outcome measure than reaction time (Kappenman et al., 2015; Reutter et al., 2017). However, the relevance of certain ERP components as indices of AB remains questionable. One study found no connection between the N2pc component and trait anxiety (Kappenman et al., 2014) . The P1 component has also been investigated, yielding inconsistent results (Carlson, 2021). Additional research is needed to corroborate and generalize these findings.

fMRI has been used to investigate neural activation patterns connected to AB in populations experiencing anxiety, depression and those who are healthy (Britton et al., 2013; Hilland et al., 2020; Monk et al., 2006; Price et al., 2014). These studies have linked AB to activation in the limbic regions, anterior cingulate cortex (ACC), and prefrontal cortex (PFC). One study using an fMRI slow event dot-probe paradigm found reduced activation in the bilateral parahippocampal/hippocampal limbic region for non-anxious participants during incongruent trials, while anxious participants showed heightened activation during the same trials. A decrease in rdACC activity was observed for both groups during incongruent trials, suggesting that anxious individuals may have more difficulty regulating limbic responses when attention is shifted away from threats (Price et al., 2014).

Research involving healthy participants performing the dot-probe task has revealed consistent activation in the ventral PFC (vPFC) and amygdala across two separate trials. The vPFC was activated when participants were exposed to 500ms of face-pair stimuli, while the amygdala was activated upon exposure to 17ms of face-pair stimuli. The study was unable to differentiate between incongruent and congruent trials (Britton et al., 2013). It has been suggested that the connectivity strength between the amygdala, ACC, and PFC is positively associated with the level of AB (Carlson et al., 2014; Carlson et al., 2013).

Additional studies have shown relationships between the amygdala and visual cortex, with correlated activity when exposed to fearful faces (Morris et al., 1996; Pessoa et al., 2002). The visual cortex has also exhibited increased activity when exposed to emotional faces during the dot-probe task (Carlson et al., 2011; Pourtois et al., 2006)

In summary, investigations using EEG and fMRI techniques have so far connected the brain's emotional attention system to the amygdala, PFC and visual cortex, with the amygdala being the primary center (Torrence & Troup, 2018).

In addition, a recent systematic review on AB recommended a multimodal approach to measuring AB to improve the reliability and validity of assessments (Carlson, 2021). By combining multiple measures, it may be possible to gain a more comprehensive understanding of AB and its underlying neural processes.

7.3 OPTICAL APPROACH USING NEAR-INFRARED LIGHT

Near-infrared (NIR) light, with wavelengths ranging from around 650 to 950 nm, can penetrate several centimeters into the head due to the low absorption by skin, skull, and brain tissue within this wavelength range (Scholkmann, 2012). Moreover, this particular spectrum is where the absorption characteristics of HbO and HbR exhibit the greatest distinction (Torricelli et al., 2014).

The journey of NIR light within tissue is complex. As photons traverse through the tissue, they interact with cellular and subcellular structures, causing the photons to scatter in random directions. This scattering phenomenon leads to multiple interactions, with photons scattering up to 10 times per centimeter of tissue (Quaresima & Ferrari, 2019). Consequently, the light that reaches the scalp, skull, and brain surface becomes scattered and weakened (Quaresima & Ferrari, 2019).

fNIRS is capable of measuring the scattered light. This is achieved by positioning a NIR laser or diode and a photodiode on the scalp, typically at a distance of around 2-4 cm apart. Upon emitting NIR light into the tissue, a portion of the light is reflected back due to scattering (Quaresima & Ferrari, 2019). The photodiode subsequently measures this reflected light. The path of the light between the laser/diode and photodiode forms a distinctive "banana-shaped" pattern, as illustrated in Figure 7-2.

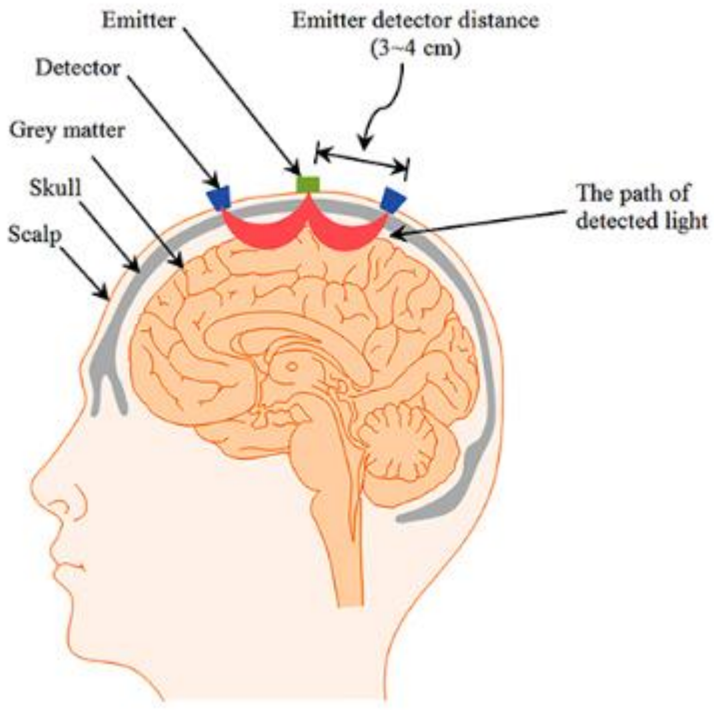

Emitter
Detector
Grey matter
Skull
Scalp
Emitter detector distance (3~4 cm)
The path of detected light

Figure 7-2 Example of emitter-detector pairs showing the "banana-shaped" paths of light, image taken from (Naseer & Hong, 2015)

The maximum depth (zmax) of the investigated tissue is determined by the distance (d) between the light source and the detector, as described by the general guideline equation: zmax = d/2 (Scholkmann & Wolf, 2012). A longer depth penetration comes at the cost of higher signal-to-noise-ratio (SNR), therefore a distance of 3 cm is often used as it reasonable compromise between SNR and depth-sensitivity (Althobaiti & Al-Naib, 2020). It is then possible to assess intensity fluctuations in the emerging light from a depth of around 1.5 cm beneath the skull while also maintaining a good SNR (Althobaiti & Al-Naib, 2020; Quaresima & Ferrari, 2019). This enables the assessment of light reflected from the cerebral cortex and the estimation of the amount of light absorbed by HbO and HbR in that specific brain region.

As a result, the intensity of the reflected light exhibits a strong correlation with the light absorption of HbO and HbR. This relationship enables fNIRS to measure alterations in HbO and HbR concentrations (Althobaiti & Al-Naib, 2020). Typically, fNIRS employs two wavelengths in the range 690-860nm, where one is more sensitive to HbO changes, and another that is sensitive to HbR [refer to figure 7-3] (Khan, Noori, et al., 2021). This dual-wavelength approach allows for the differentiation of absorption between the two chromophores.

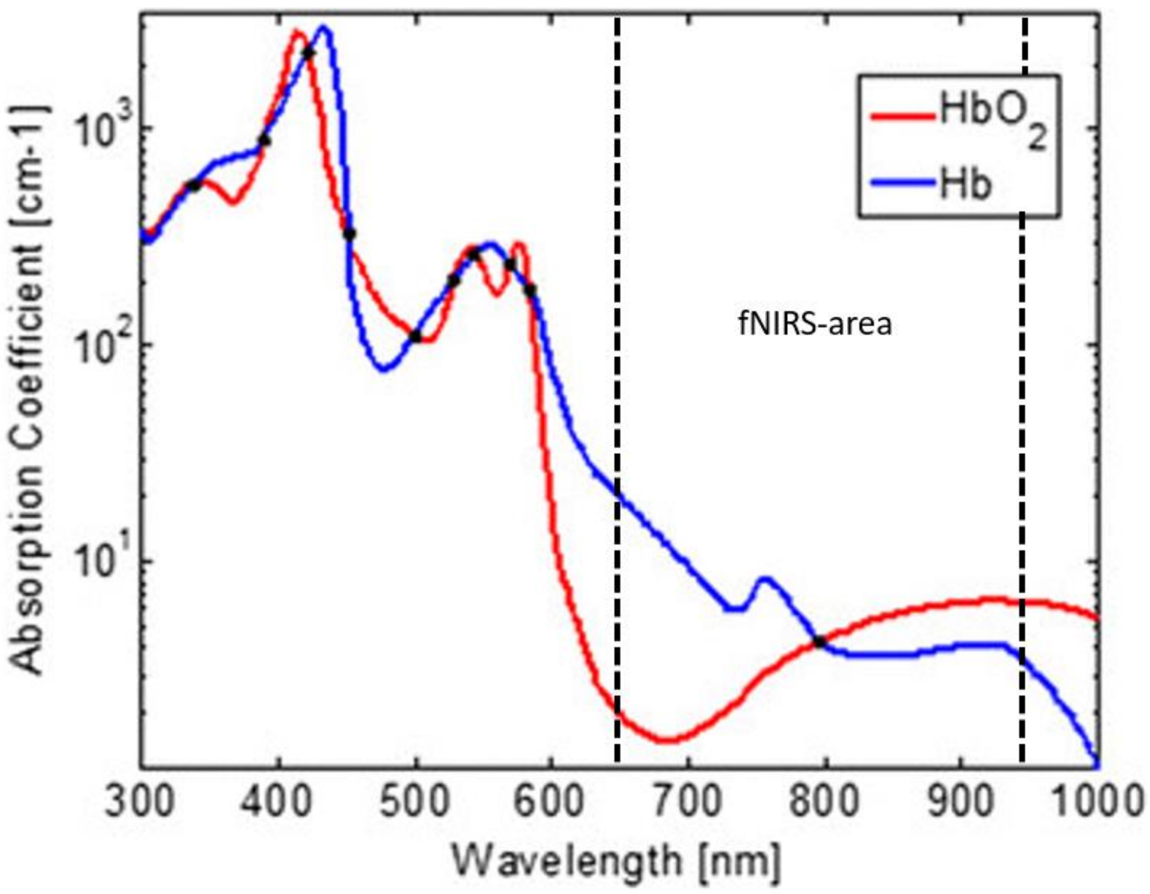


Figure 7-3 Ilustration of the absorption spectra of HbO (red) and HbR (blue) with respect to light wavelength. It is evident that HbR exhibits a significantly higher absorption rate compared to HbO at around 700nm, whereas the opposite is true at around 850nm. The image source is (Liu et al., 2015).

There are three main fNIRS techniques, each based on a specific type of illumination: continuous wave (CW), frequency-domain (FD), and time-domain (TD), with each having its own advantages and disadvantages:

1. Continuous Wave (CW) fNIRS: This technique uses constant tissue illumination and measures the attenuation of light as it passes through the head. CW-based systems are low cost and easily transportable, making them an accessible

option for researchers. Changes in HbO and HbR concentrations are measured using a modification of the Lambert-Beer's law. While this method is affordable and portable, it only provides relative changes in absorption, and cannot differentiate between absorbed and scattered light (Quaresima & Ferrari, 2019; Scholkmann & Wolf, 2012).

2. Frequency-Domain (FD) fNIRS: In this technique, the head is illuminated with intensity-modulated light, and both the attenuation and phase delay of the emerging light are measured. FD fNIRS provides more accurate and detailed measurements than the CW modality, as it can distinguish between absorbed and scattered light. However, this method is more complex and expensive compared to CW fNIRS, which may limit its accessibility (Quaresima & Ferrari, 2019; Scholkmann & Wolf, 2012).

3. Time-Domain (TD) fNIRS: This method involves illuminating the head with short pulses of light and detecting the shape of the pulse after it propagates through tissues. TD fNIRS offers the most precise measurements of HbO and HbR concentrations, enabling a high level of accuracy in brain activity measurements. Despite its precision, TD fNIRS is also the most complex and expensive of the three techniques, and has a lower sampling rate compared to the other two methods (Quaresima & Ferrari, 2019; Scholkmann & Wolf, 2012).

Out of these three, the CW technique is the most commercially available and cost-effective, and as a result, the one that has used the most in research (Quaresima & Ferrari, 2019).

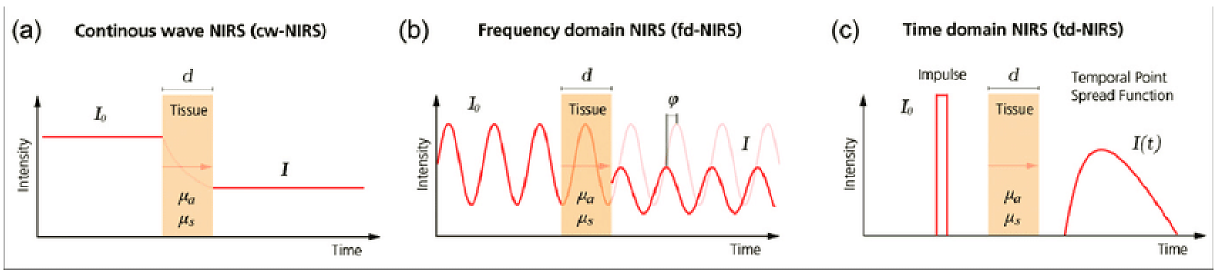


Figure 7-4 Illustration the three types fNIRS techniques, picture taken from (Scholkmann, 2012).

In these cognitive studies, the results from neurovascular coupling are often the signal of interest (Phillips et al., 2016). When nerve cells become active during tasks like thinking or sensing, they release chemical messengers called neurotransmitters (e.g., glutamate). As the neurons become more active, they need more energy in the form of oxygen and nutrients. This increased demand leads to higher oxygen consumption in the brain (Krishnamoorthy-Natarajan & Koide, 2016).

To meet this energy demand, our brain increases local cerebral blood flow (CBF) to deliver more oxygen. Neurotransmitters activate specific receptors on star-shaped cells called astrocytes. This causes a chain reaction within the astrocytes, increasing calcium levels in their branch-like extensions (endfeet) that wrap around small blood vessels in the brain (Krishnamoorthy-Natarajan & Koide, 2016).

The increase in calcium levels in the astrocyte endfeet causes the nearby small blood vessels to widen, allowing more blood to flow through the brain. This increase in blood flow delivers more oxygen and nutrients to the active nerve cells (Krishnamoorthy-Natarajan & Koide, 2016). During this process, the levels of oxygen-carrying molecules in the blood change. There is an increase in oxygenated HbO and a smaller decrease in HbR. As a result, the total amount of hemoglobin (HbT) in the blood increases (Lloyd-Fox et al., 2010).

Since neuronal activity is correlated with these hemodynamic changes, fNIRS can be used to measure brain activity by detecting these changes in blood oxygenation (Reddy et al., 2021).

While research on AB using fNIRS is limited, one investigation, using a CW-fNIRS equipment with 8 sources (690 nm and 830 nm wavelengts) and 9 detectors mounted at PFC region, observed alterations in HbO concentrations in both the medial PFC and bilateral PFC during congruent and incongruent trials in a dot-probe task involving fearful faces (Torrence, 2015). This finding implies that fNIRS might be a valuable tool for examining AB.

8 METHODS

To ensure a successful experimental process, we began by setting up the test lab, which involves installing all the necessary hardware and software components. Once the lab infrastructure was in place, we proceeded to design the experiment, carefully outlining the objectives, variables, and controls. To facilitate smooth execution, a step-by-step guide detailing the experimental procedures will be prepared.

Before conducting the actual experiment, we did a test-run to verify the functionality of the hardware and software, as well as the intuitiveness of the lab procedures. This test-run helped identify any potential issues, enabling us to make improvements based on the findings. By refining the lab setup and procedures, we aimed to create a robust and user-friendly pipeline that will promote efficiency, reliability, and repeatability in the experimental process.

We couldn't set up a hybrid EEG/ET and hybrid fNIRS/ET in the beginning as it was not possible to move the ET to the same lab as the EEG and fNIRS equipment was. Consequently, for the test-run, we simply conducted each imaging method separately. Later, we acquired an ET for the lab, and were then able to create the hybrid setups for the experiment.

8.1 SETTING UP THE LAB

Our research team has been granted permission to establish a laboratory in the Medtech.testlab, located in building P35 at OsloMet University. In anticipation of relocating the ET to the Medtech.testlab for the experiment, we purposefully designed the lab to cater to this requirement. Within this lab, we have set up both the EEG and fNIRS systems. Additionally, we have installed a dedicated computer with all necessary software installed. This computer is connected via an HDMI cable and a USB cable to a 53x30cm 1080p monitor and keyboard on the left side of the lab, where participants will be seated during the dot-probe task. The ET was set up at the interaction Lab in building P35 at OsloMet University.

The fNIRS equipment we installed is the **NIRScout** system, manufactured by NIRx (Berlin, Germany). This equipment utilizes CW technology to measure Hb levels.

This system applies two wavelengths of 760nm and 850nm with optical fibres. Our experimental setup employs a 42-channel, 16x16 prefrontal and occipital cortex montage, adhering to the standard 10/20 arrangement [See figure 8-1]. The sampling rate for our study is set at initially set to 3.91 Hz after doing the test-run (it was set to 7.81 Hz before the test-run). The probes on the NIRScout are wired, we have therefore attached a cable holder to the table to alleviate the weight of the wires on the participant's head, ensuring their comfort.

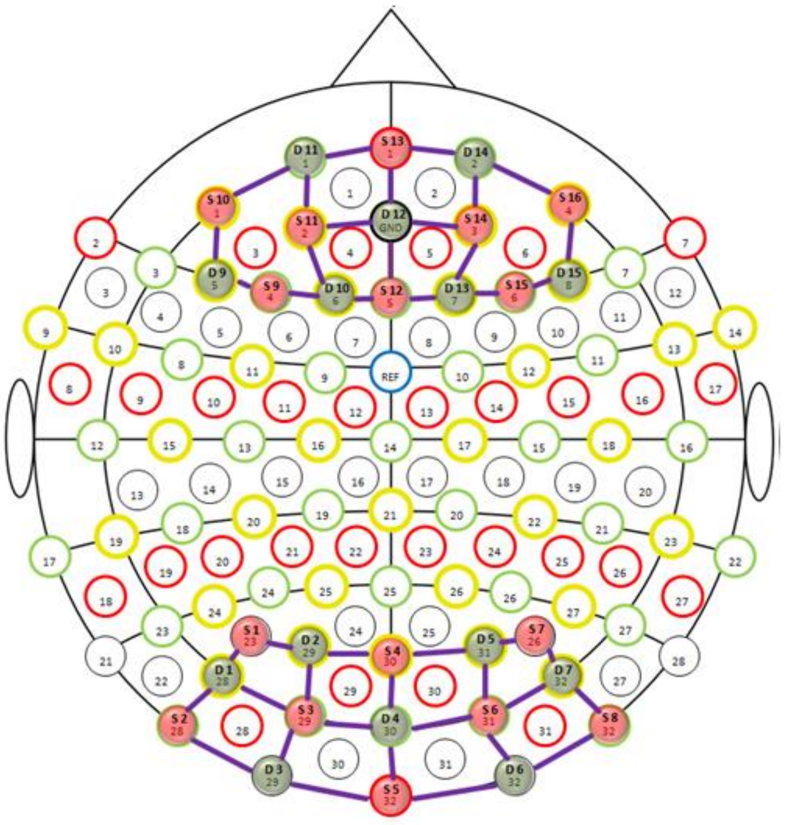


Figure 8-1 A visualization of our montage configuration, with LEDs as red and photodiodes as green, as depicted in the NIRStar 15-3 software interface.

The **NIRStar 15-3** software, developed by NIRx, is included with the NIRScout system. This software not only calibrates the equipment but also gathers the fNIRS data. NIRStar offers a user-friendly interface that enables customization of the sampling rate, cap montage setup, participant age input, and trigger connection establishment. It also features real-time fNIRS signal monitoring and stores data in its raw format. As a specialized software for fNIRS systems, NIRStar serves as a comprehensive solution for our data collection requirements.

The EEG equipment we setup is the **g.Nautilus 32 channel** system by g.tec medical engineering, which features 32 active electrodes and a medium-sized cap (54-58cm) with a sampling rate of 500Hz. The cap montage setup is based on the widely used 10/20 international standard for EEG research and clinical settings. The highest accepted impedance is 25 ohm, enabling us to accurately capture the electrical activity in the brain with minimal noise.

The system is delivered with the **g.Recorder Version 1.20.03 software**, which we are using for the recording of EEG data. The software is user friendly and highly customizable, enabling us to configure the system to our specific needs. It includes a range of user-defined tools to ensure the precise and accurate acquisition and recording of data, such as support for different sampling rates, filtering, and montage configurations.

The ET system we setup is the **EyeLink Portable Duo** by SR Research, which is designed to capture precise eye movement data. This device features a high-speed infrared camera and a Host-PC, which is a laptop equipped with a dedicated operating system tailored for the ET. The operating system is optimized to achieve exceptional temporal precision. To ensure the accuracy of our data collection, we utilize a sampling rate of 1000 Hz and the provided head mount to minimize participant movement during the experiment.

To ensure consistent environmental conditions for all participants, we paid close attention to factors such as sound and lighting. For instance, one study demonstrated that light conditions can significantly impact task performance, response time, and accuracy (Yuan et al., 2021). Moreover, changes in lighting can affect the ET and fNIRS signals. Consequently, we have installed lightproof curtains to maintain a consistent level of light for each participant. We also included a light measurer, to further make sure the lighting conditions was the same for each participant.

We also attempted to minimize noise from external sounds. We ensured that the air conditioning unit was switched off during the experiment and placed posters outside the test lab to remind passers-by to remain quiet while an experiment was in progress. Although we could not fully eliminate all external noise, the sound of the tram will be barely audible and infrequent. Also, a wall has been set up between the researcher and participant areas to prevent any distraction during the testing process.

Due to the EEG's sensitivity to electronic devices (Usakli, 2010), we have positioned the computer and other devices at least 1.5 meters away from the participant's testing area (excluding the stimulus screen and keyboard).

8.2 EXPERIMENT PARADIGM DESIGN

When conducting a study concerning cognitive neuroscience, there is often a need for a stimulus program. The stimulus program has two roles: Presenting stimuli to the participant and communicating what and when stimuli is presented to the recording-program (more on this under the time and signal synchronization section).

Psychopy is a free open-source stimulus program that uses the python coding language. There are two ways to create experiment paradigm with Psychopy: using pure coding or with the builder, which is a graphical user interface (GUI). For this experiment, Psychopy-Builder was used. The reason is that it is easy to learn and simultaneously able to create complex and precise paradigms. Psychopy-Builder is also able to generate a complete pyton code of the experiment, which can be edited manually.

The way an experiment paradigm is created is by first setting up routines [see figure 8-2]. It is then possible to add different components to each routine. It could be stimulus components (e.g., image, sound, text) or response components (e.g., keyboard pressed, mouse clicked). There is also a code component where custom python codes can be written. All these components are highly customizable (e.g., timing, what to present). A loop function can be added if it is required for some of the routines to run multiple times.

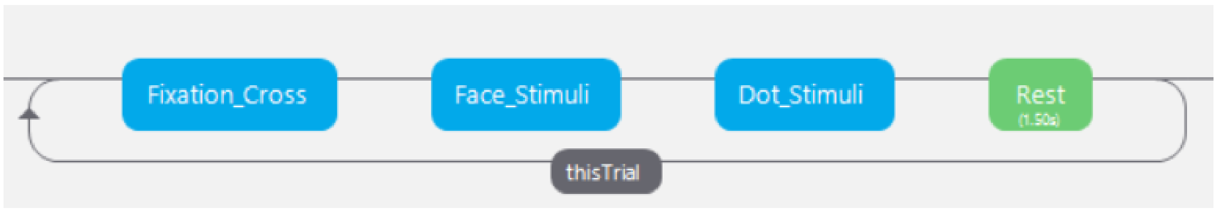


Figure 8-2 Illustration of Routines and Loops in a Dot-Probe Task. Each routine is depicted as a rectangular box, with the program sequentially executing them from left to right. A single pass through all four boxes constitutes a trial. The loop is represented by the 'thisTrial' box, allowing the user to specify the desired number of trial repetitions. Image sourced from a dot-probe task created using PsychoPy.

The experiment paradigm designed is a dot-probe task. There are many variants of the dot-probe-task that uses different stimulus present time (between 17ms and up to 2000ms) and different kind of stimuli that is presented (words, faces, colours). The paradigm used in this experiment is almost a replica of the dot-probe-task used in this study (Andrzejewski & Carlson, 2020), which uses face pairs (fearful vs. neutral) with different facial emotions as stimuli and a 200ms stimulus present time. The following changes were implemented: elimination of sounds and addition of pairs of happy vs. neutral facial expressions.

The dot-probe-task for EEG and eye tracking are identical. The only thing that is different for fNIRS is the use of a slow-event-design, which means a longer rest-time in between each trail. This is due to the hemodynamic response having lower temporal characteristics (Meryem A. Yücel, 2021). We employed the same rest-time of 7 seconds for our fNIRS paradigm, as utilized in the fNIRS dot-probe experiment conducted by (Torrence, 2015).

Here is an explanation of the paradigm:

The participant is first presented with a text that tells them to press a key when they are ready, this will start a 30s timer which appears on the screen. This is done to reduce stress in the participant before the task starts.

After 30s has passed, the dot-probe task will begin. First a fixation cross will appear for 1000ms. This is done to make the participant focus on the middle of the screen.

After that, two emotional faces will appear on the screen (one on each side) for 200ms. There are three types of face pairs: neutral/neutral, happy/neutral, fearful/neutral (figure 8-3 shows a fearful/neutral pair).

Then, a dot will appear on one side of the screen. There are two types of tasks: Congruent, which means that the dot is behind the most negative emotional face (e.g.: behind the neutral face if the face pair is happy/neutral). Or Incongruent, which means that the dot is behind the most positive emotional face [See figure 8-3].

The participant has been instructed beforehand to push ‘z’ if dot appears on left side and ‘m’ if on right side. When one of the keys is pressed, the program goes into rest/pause and the screen will turn black for 1000ms (7000ms for fNIRS).

Then, the task starts over again with a new set of face pairs. This is repeated 160 times (80 times for fNIRS due to longer rest).

When 160 (80) trails have passed, the text “Experiment is over, thank you” will show up on the screen, before closing the program. The total experiment will last approximately 10 minutes.

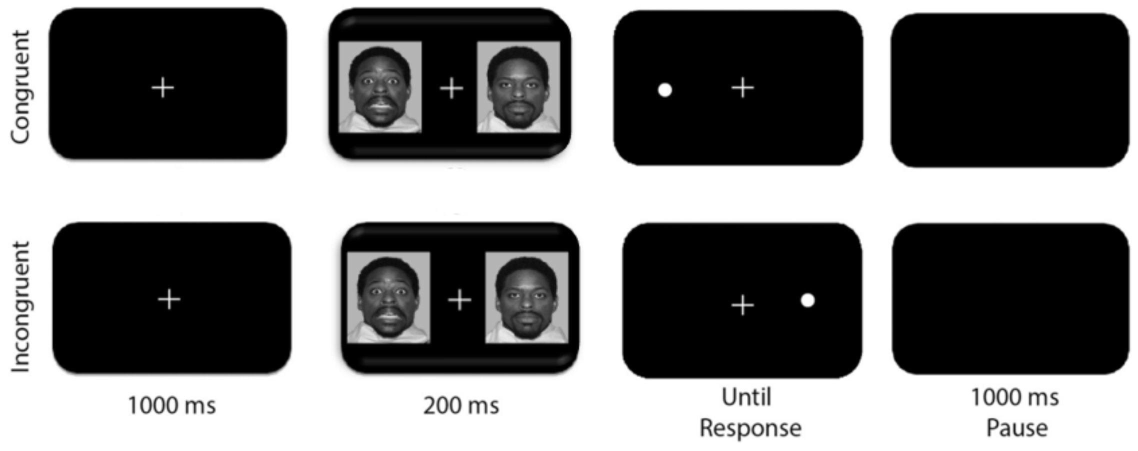


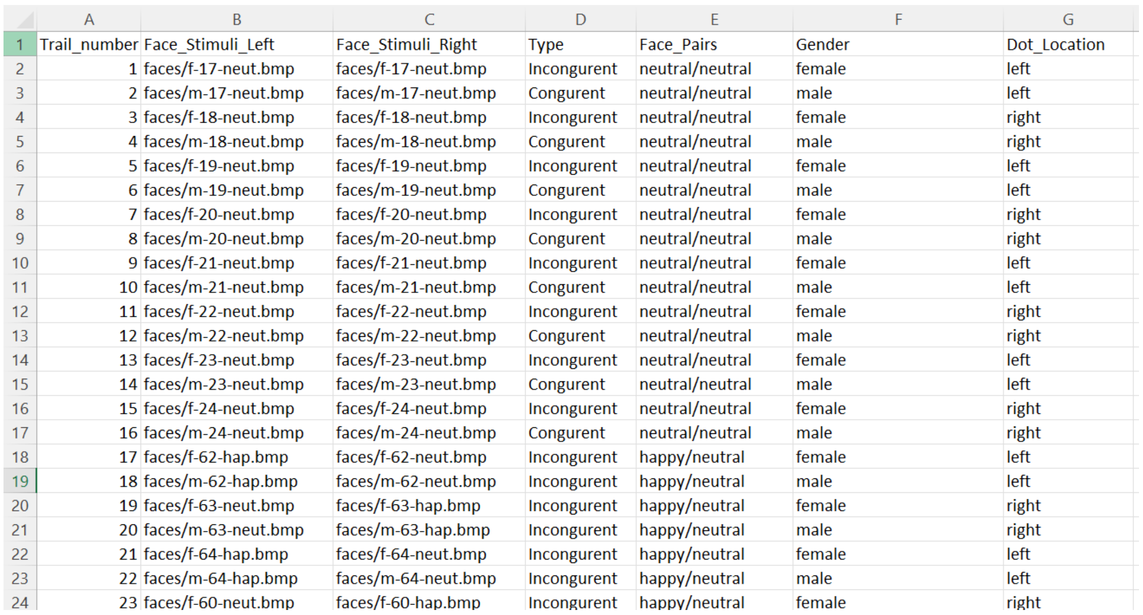
Figure 8-3 Overview of the dot-probe task, picture taken from (Andrzejewski & Carlson, 2020) with some edits to fit our setup.

To accommodate the differences between each imaging modality (EEG, fNIRS, ET), we have created an individual dot-probe-task for each of them. This was necessary because each modality employs a different recording program, requiring adjustments to establish a connection between Psychopy (stimulus program) and the respective recorder program to send triggers.

The task was designed to be as similar as possible across all image-modalities, to enable comparisons and identify potential correlations between fNIRS, EEG, and ET results. By using the same stimulus program (Psychopy) for all modalities, we ensured consistency in delays and "errors", making it easier to use a hybrid approach in the future.

An Excel document was created to contain the various conditions for each trial. These conditions specify the type of images to be presented, the location of the dot, and other relevant variables. There are a total of 80 different conditions, with each row representing one condition [refer to Figure 8-4]. Each time the dot-probe trial is repeated, a row is randomly selected from the Excel document, and its values are assigned to the variables in the program (e.g., Dot_Location = left). The selection process ensures that all conditions are used once before any are repeated. The EEG and ET modalities undergo all 80 conditions twice, resulting in a total of 160 trials, while fNIRS only undergoes them once, resulting in a total of 80 trials. This is because fNIRS requires a longer rest time between each trial and repeating all 80 conditions twice would make the experiment excessively long.

Randomization further strengthens the experiment by minimizing the impact of confounding factors, such as the learning effect, which could otherwise lead to misleading conclusions. By randomly selecting a condition from the Excel document for each trial, the experiment prevents systematic improvement in performance due to increased familiarity or practice with the task, or so called ‘practice-effect’ (Duff et al., 2007).



	A	B	C	D	E	F	G
1	Trail_number	Face_Stimuli_Left	Face_Stimuli_Right	Type	Face_Pairs	Gender	Dot_Location
2	1	faces/f-17-neut.bmp	faces/f-17-neut.bmp	Incongruent	neutral/neutral	female	left
3	2	faces/m-17-neut.bmp	faces/m-17-neut.bmp	Congruent	neutral/neutral	male	left
4	3	faces/f-18-neut.bmp	faces/f-18-neut.bmp	Incongruent	neutral/neutral	female	right
5	4	faces/m-18-neut.bmp	faces/m-18-neut.bmp	Congruent	neutral/neutral	male	right
6	5	faces/f-19-neut.bmp	faces/f-19-neut.bmp	Incongruent	neutral/neutral	female	left
7	6	faces/m-19-neut.bmp	faces/m-19-neut.bmp	Congruent	neutral/neutral	male	left
8	7	faces/f-20-neut.bmp	faces/f-20-neut.bmp	Incongruent	neutral/neutral	female	right
9	8	faces/m-20-neut.bmp	faces/m-20-neut.bmp	Congruent	neutral/neutral	male	right
10	9	faces/f-21-neut.bmp	faces/f-21-neut.bmp	Incongruent	neutral/neutral	female	left
11	10	faces/m-21-neut.bmp	faces/m-21-neut.bmp	Congruent	neutral/neutral	male	left
12	11	faces/f-22-neut.bmp	faces/f-22-neut.bmp	Incongruent	neutral/neutral	female	right
13	12	faces/m-22-neut.bmp	faces/m-22-neut.bmp	Congruent	neutral/neutral	male	right
14	13	faces/f-23-neut.bmp	faces/f-23-neut.bmp	Incongruent	neutral/neutral	female	left
15	14	faces/m-23-neut.bmp	faces/m-23-neut.bmp	Congruent	neutral/neutral	male	left
16	15	faces/f-24-neut.bmp	faces/f-24-neut.bmp	Incongruent	neutral/neutral	female	right
17	16	faces/m-24-neut.bmp	faces/m-24-neut.bmp	Congruent	neutral/neutral	male	right
18	17	faces/f-62-hap.bmp	faces/f-62-neut.bmp	Incongruent	happy/neutral	female	left
19	18	faces/m-62-hap.bmp	faces/m-62-neut.bmp	Incongruent	happy/neutral	male	left
20	19	faces/f-63-neut.bmp	faces/f-63-hap.bmp	Incongruent	happy/neutral	female	right
21	20	faces/m-63-neut.bmp	faces/m-63-hap.bmp	Incongruent	happy/neutral	male	right
22	21	faces/f-64-hap.bmp	faces/f-64-neut.bmp	Incongruent	happy/neutral	female	left
23	22	faces/m-64-hap.bmp	faces/m-64-neut.bmp	Incongruent	happy/neutral	male	left
24	23	faces/f-60-neut.bmp	faces/f-60-hap.bmp	Incongruent	happy/neutral	female	right

Figure 8-4 An Excel file containing PsychoPy variables, as sourced from an Excel spreadsheet.

The conditions are divided like this:

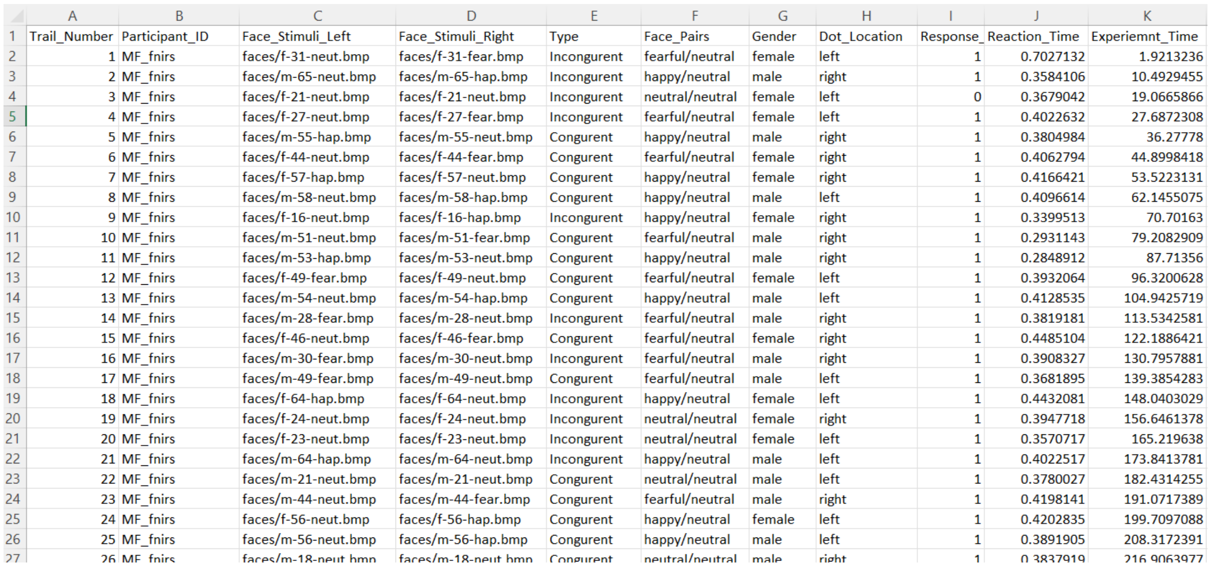
- Half of the trials are split male / female.
- Half of the trials are split Incongruent / Congruent.
- 32 is Happy/Neutral, 32 is Fearful/Neutral and 16 is Neutral/Neutral.
- Image pairs shown is always the same person on each side.
- There is a different persons face on each trial (In total 80 different people is included).

The experimental design incorporates several elements to enhance its validity and control for potential confounds. The same person's face is displayed in each image pair to ensure that no individual automatically draws more attention due to more pronounced facial features. This approach helps maintain a consistent level of attention across all trials. The experiment also ensures an equal representation of male and female faces, as studies have indicated the importance of gender balance. For instance, one study found that adult male participants tend to allocate more attention to female faces compared to male faces (Okazaki et al., 2010).

Incorporating 80 unique individuals faces in the experiment mitigates the 'learning' effect that may arise from repeated exposure to the same stimulus. This approach is supported by findings from a previous study, which demonstrated that responses to the dot probe were faster when it emerged in a location signalled by a predictive stimulus, as opposed to a location indicated by a nonpredictive stimulus (Mogg et al., 2007).

Incorporating neutral-neutral face pair trials, the study draws inspiration from an adapted dot-probe task (Pfabigan et al., 2014). These baseline trials serve as a reference point, to better comprehend the influence of emotional stimuli on the results. The decision to use happy stimuli is based on studies that highlight their connection to depression (Thomas Armstrong & Bunmi O. Olatunji, 2012). By incorporating happy stimuli, the experiment can explore AB associated with mood disorders. Fearful faces are chosen over angry ones, as they have been demonstrated to produce similar dot-probe RT and ET biases (Mogg et al., 2007) and hold broader implications for fear perception. Fearful faces suggest a more general, undefined threat, while angry faces may be perceived as a threat specific to social situations (Price et al., 2014). This selection ensures a more comprehensive examination of AB related to various forms of threat.

Upon completion of the experiment, a new Excel document is generated, which includes each trial condition along with reaction time and whether the participant pressed the correct button [see figure 8-5]. This is done to incorporate reaction time into the analysis and provide a comprehensive overview of the conditions presented in each specific trial. Consequently, the researcher can easily examine which images were displayed on the right or left, the location of the dot, gender of face stimulus and whether the participant responded correctly. This gives the opportunity to later do a more in-depth evaluation of the experiment's results.



	A	B	C	D	E	F	G	H	I	J	K
1	Trail_Number	Participant_ID	Face_Stimuli_Left	Face_Stimuli_Right	Type	Face_Pairs	Gender	Dot_Location	Response_	Reaction_Time	Experiemnt_Time
2	1	MF_fnirs	faces/f-31-neut.bmp	faces/f-31-fear.bmp	Incongurent	fearful/neutral	female	left	1	0.7027132	1.9213236
3	2	MF_fnirs	faces/m-65-neut.bmp	faces/m-65-hap.bmp	Incongurent	happy/neutral	male	right	1	0.3584106	10.4929455
4	3	MF_fnirs	faces/f-21-neut.bmp	faces/f-21-neut.bmp	Incongurent	neutral/neutral	female	left	0	0.3679042	19.0665866
5	4	MF_fnirs	faces/f-27-neut.bmp	faces/f-27-fear.bmp	Incongurent	fearful/neutral	female	left	1	0.4022632	27.6872308
6	5	MF_fnirs	faces/m-55-hap.bmp	faces/m-55-neut.bmp	Congurent	happy/neutral	male	right	1	0.3804984	36.27778
7	6	MF_fnirs	faces/f-44-neut.bmp	faces/f-44-fear.bmp	Congurent	fearful/neutral	female	right	1	0.4062794	44.8998418
8	7	MF_fnirs	faces/f-57-hap.bmp	faces/f-57-neut.bmp	Congurent	happy/neutral	female	right	1	0.4166421	53.5223131
9	8	MF_fnirs	faces/m-58-neut.bmp	faces/m-58-hap.bmp	Congurent	happy/neutral	male	left	1	0.4096614	62.1455075
10	9	MF_fnirs	faces/f-16-neut.bmp	faces/f-16-hap.bmp	Incongurent	happy/neutral	female	right	1	0.3399513	70.70163
11	10	MF_fnirs	faces/m-51-neut.bmp	faces/m-51-fear.bmp	Congurent	fearful/neutral	male	right	1	0.2931143	79.2082909
12	11	MF_fnirs	faces/m-53-hap.bmp	faces/m-53-neut.bmp	Congurent	happy/neutral	male	right	1	0.2848912	87.71356
13	12	MF_fnirs	faces/f-49-fear.bmp	faces/f-49-neut.bmp	Congurent	fearful/neutral	female	left	1	0.3932064	96.3200628
14	13	MF_fnirs	faces/m-54-neut.bmp	faces/m-54-hap.bmp	Congurent	happy/neutral	male	left	1	0.4128535	104.9425719
15	14	MF_fnirs	faces/m-28-fear.bmp	faces/m-28-neut.bmp	Incongurent	fearful/neutral	male	right	1	0.3819181	113.5342581
16	15	MF_fnirs	faces/f-46-neut.bmp	faces/f-46-fear.bmp	Congurent	fearful/neutral	female	right	1	0.4485104	122.1886421
17	16	MF_fnirs	faces/m-30-fear.bmp	faces/m-30-neut.bmp	Incongurent	fearful/neutral	male	right	1	0.3908327	130.7957881
18	17	MF_fnirs	faces/m-49-fear.bmp	faces/m-49-neut.bmp	Congurent	fearful/neutral	male	left	1	0.3681895	139.3854283
19	18	MF_fnirs	faces/f-64-hap.bmp	faces/f-64-neut.bmp	Incongurent	happy/neutral	female	left	1	0.4432081	148.0403029
20	19	MF_fnirs	faces/f-24-neut.bmp	faces/f-24-neut.bmp	Incongurent	neutral/neutral	female	right	1	0.3947718	156.6461378
21	20	MF_fnirs	faces/f-23-neut.bmp	faces/f-23-neut.bmp	Incongurent	neutral/neutral	female	left	1	0.3570717	165.219638
22	21	MF_fnirs	faces/m-64-hap.bmp	faces/m-64-neut.bmp	Incongurent	happy/neutral	male	left	1	0.4022517	173.8413781
23	22	MF_fnirs	faces/m-21-neut.bmp	faces/m-21-neut.bmp	Congurent	neutral/neutral	male	left	1	0.3780027	182.4314255
24	23	MF_fnirs	faces/m-44-neut.bmp	faces/m-44-fear.bmp	Congurent	fearful/neutral	male	right	1	0.4198141	191.0717389
25	24	MF_fnirs	faces/f-56-neut.bmp	faces/f-56-hap.bmp	Congurent	happy/neutral	female	left	1	0.4202835	199.7097088
26	25	MF_fnirs	faces/m-56-neut.bmp	faces/m-56-hap.bmp	Congurent	happy/neutral	male	left	1	0.3891905	208.3172391
27	26	MF_fnirs	faces/m-18-neut.bmp	faces/m-18-neut.bmp	Congurent	neutral/neutral	male	right	1	0.3837919	216.9063977

Figure 8-5 An Excel list produced by PsychoPy, as extracted from an Excel spreadsheet.

8.2.1 Time and signal synchronization

During the analysis of data, it is crucial to determine the type of stimulus presented to the participants at any given time. This is typically achieved by sending a trigger signal, which can either be manually generated within the recording program or automatically generated by a stimulus program. In this experiment, triggers were exclusively sent from the stimulus program to ensure maximum precision. Given the high temporal resolution of both EEG and ET technologies, which can detect changes occurring over a few milliseconds, minimizing the delay between stimulus presentation and trigger signal is essential. Although fNIRS has a lower sampling rate, it's still important to keep the trigger delay as low as possible, even though it may not be as critical as in other methods. The stippled line in Figure 8-6 provides an example of how a trigger signal appears during signal acquisition.

It is worth noting that different trigger signals can be used, with both the EEG and fNIRS systems allowing for up to eight different triggers, while the ET system has the capability to receive an infinite number of triggers.

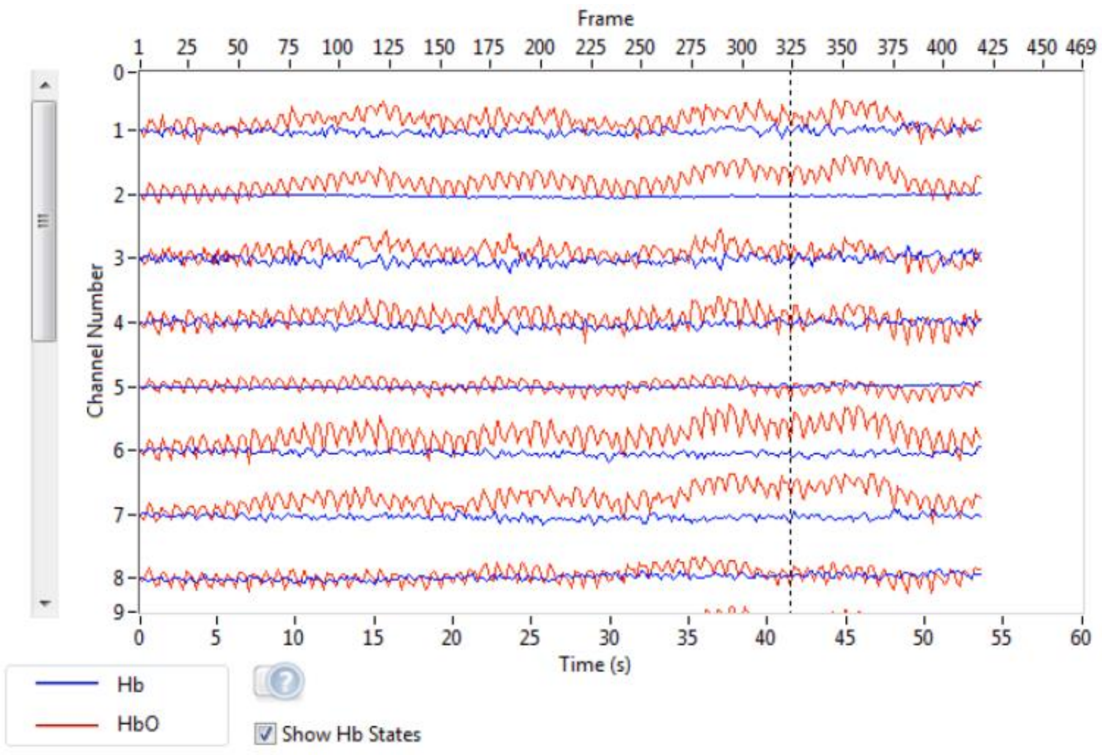


Figure 8-6 Exemplifies a trigger event (stippled line) in NirStar 15-3, as presented in the NIRx Trigger Manual (NIRx, 2019b).

Each image modality requires different ways to setup connection and send triggers:

nirScout (fNIRS) uses lab-stream-layer (LSL), which establishes a digital connection between the two softwares NirsStar 15-3 and Psychopy. This is done by adding this code at the start of the experiment using the code-component in Psychopy-Builder:

from pylsl import StreamInfo, StreamOutlet # import required classes

info = StreamInfo('TriggerStream', type='Markers', channel_count=1, channel_format='int32', source_id='Example') # sets variables for object info

outlet = StreamOutlet(info)

It is then possible to send triggers with the code component using this code (x can range from 1 to 8): outlet.push_sample([x]).

g.Nautulus (EEG) does not need any additional code since it is using a hardware connection via a parallel port. Psychopy has a built-in parallel port component in the Psychopy-builder. Here triggers are sent using binary whole binary numbers (1, 2, 4, 16, 32, 64, 128).

EyeLink Portable Duo (ET) need much more code in Psychopy to connect to the Host-PC (See A.4 for code). This code is added in a code component at the start of the Psychopy experiment. It is then possible to send triggers with this code: el_tracker.sendMessage('trigger'). The ET can receive whole sentences as triggers.

Stimulus	fNIRS	EEG	ET
Experiment Start/Stop	1	1	'Experiemtn_Start' / '_Stop'
Fixation Cross	2	2	'Fixation_Cross_Start' / '_Stop'
Face Pairs: Neutral/Neutral	3	4	'Face_Pairs_Neutral_Neutral_Start' / '_Stop'
Face Pairs: Happy/Neutral	4	8	'Face_Pairs_Happy_Neutral_Start'/ '_Stop'
Face Pairs: Fearful/Neutral	5	16	'Face_Pairs_Fearful_Neutral_Start' / '_Stop'
Dot Congruent	6	32	'Dot_Congruent_Start' / '_Stop'
Dot Incongruent	7	64	'Dot_Incongruent_Start' / '_Stop'
Reaction (keyboard pushed)	8	128	'Reaction'

Table 8-1 An overview of the relationship between trigger signals and stimuli. It is important to note that the complete commands for the fNIRS and ET are not displayed in this figure, specifically outlet.push_sample([x]) for fNIRS and el_tracker.sendMessage('trigger') for the ET. The triggers for EEG and fNIRS are only sent at the beginning of each stimulus, whereas the ET sends triggers for both the start and end of each stimulus.

8.3 Experiment Protocol

8.3.1 Step-by-step guide

A step-by-step guide for all image modalities was developed to ensure a seamless experiment process for participants. This guide is essential due to the numerous details to recall, and the possibility of overlooking crucial elements that could jeopardize the experiment. Furthermore, other students will be trained to assist with the experiment, necessitating a well-structured lab pipeline for current and future experiments. Refer to Appendix 1 for the complete guide; however, note that it is the refined and finalized version and may not precisely align with the test-run version.

8.3.2 Experiment Order

The experiment maintains a fixed sequence for experimental measurements: ET, fNIRS, and EEG. While randomization could enhance experimental control, as suggested by (Suresh, 2011), it also presents practical challenges:

After EEG measurements, participants are advised to shower for comfort, which would require disassembling and cleaning the fNIRS cap after each participant. If participants shower between measurements, it could prolong the experiment and potentially induce fatigue during subsequent assessments. By positioning EEG as the final measurement, participants can wash their hair after the experiment without time constraints or shower at home if preferred.

ET measurements took place in a separate laboratory, and scheduling these as a second measure would require additional travel time for participants, increasing the experiment's duration and potential fatigue.

Given these considerations, randomizing the measurements was deemed counterproductive, as it could increase the experiment's length and potentially lead to participant fatigue. Instead, a fixed order was determined to be the most time-efficient and comfortable for participants: ET first, followed by fNIRS, and finally EEG.

8.3.3 Procedure for each modality

To ensure the correct cap size is used for the fNIRS measurement, the circumference of the participant's head is first measured. This is done by measuring around the head from the Nasion, over the ears, and to the Inion [see figure 8-7]. There are three different cap sizes to choose from (54cm, 56cm, and 58cm), and the appropriate size is selected by rounding down the head circumference measurement. For example, if the head circumference is 55.3cm, a cap size of 54cm will be used. This ensures that the probes do not hang loosely while still providing comfort to the participant.

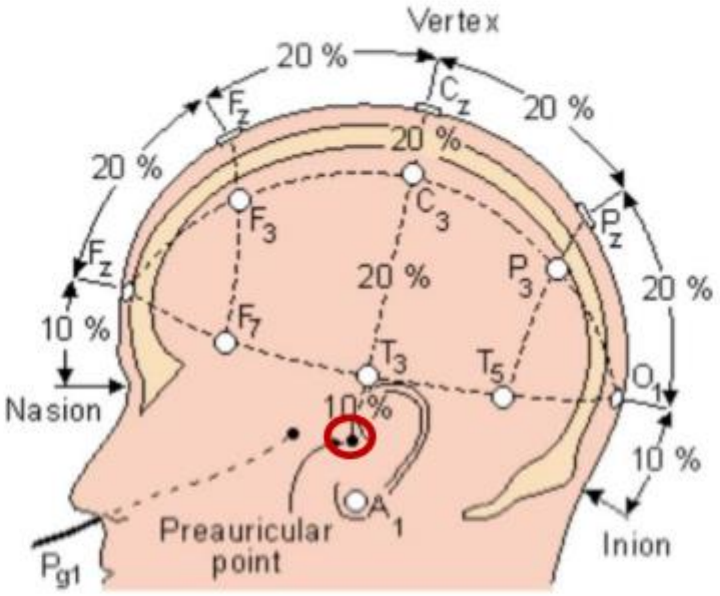


Figure 8-7 Displays the head coordinate system utilized by both EEG and fNIRS setups, as depicted in the NIRx NIRSCap User Guide (NIRx, 2019).

Once the correct cap size is selected, the participant is positioned approximately 70 cm away from the computer screen. The cap is then placed on the participant's head, ensuring proper probe placement and skin contact. The distance between the Nasion and the first probe (Fz) should be 3cm, and the probes in the center (Cz) should align with the middle of the head. The cap is fastened with a strap under the chin to prevent movement during the experiment.

To achieve proper contact between the LED/photodiode (emitter/detector) and the skin, any hairs obstructing the probes are pushed away using a Q-tip. The goal is to remove any potential obstacles that may interfere with the signal. A light-proof hood is then placed over the fNIRS cap to prevent noise from external light sources.

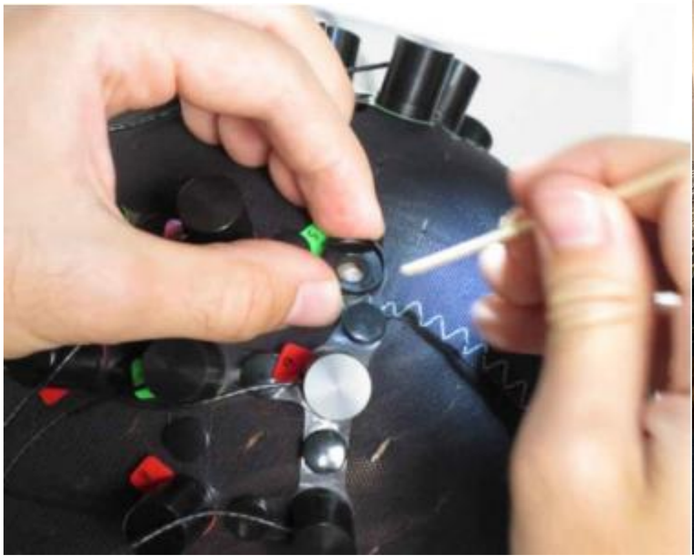
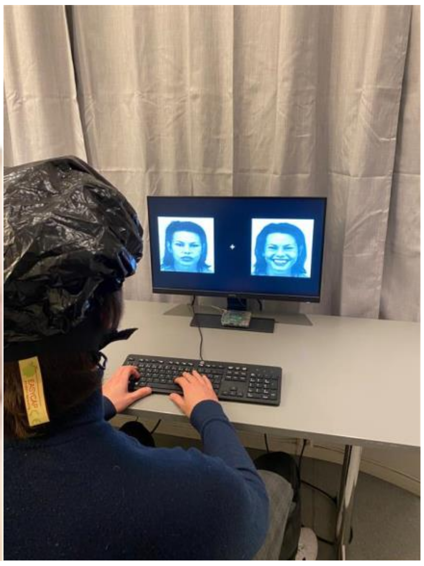



Figure 8-8 (left) Demonstrates the technique for utilizing a q-tip to clear away hair, as presented in the NIRx Troubleshooting Signal Quality Getting Started Guide (NIRx, 2017).

Figure 8-9 (right) Depicts a participant engaged in the dot-probe task, wearing the fNIRS cap with shower-cap, captured during the test-run.

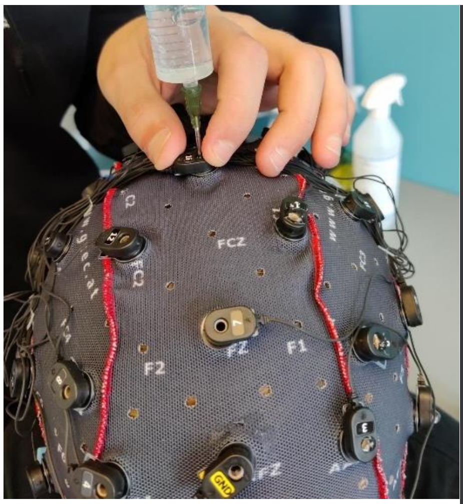
Calibration is carried out via the NIRStar 15-3 software, which checks for signal sufficiency. During calibration, diodes touching the skin should result in excellent contact, but sometimes this may not be the case due to factors such as poor contact, dark skin, thick hair, or the participant moving the cap. Calibration results are indicated by color codes: white for no contact, red for poor contact, yellow for ok contact, and green for excellent contact. If necessary, probes can be removed and repositioned with a Q-tip to improve the signal.

The aim is to achieve as many green signals as possible, but this can be challenging for some participants. Therefore, a maximum time of 35 minutes is allotted for calibration. If calibration takes longer than 35 minutes, all channels that are not green are noted in the log.

To ascertain the appropriate positioning of the EEG cap, the participant's head mid-point (Cz) must first be determined. This involves measuring the distance between the pre-auricular points (ear-to-ear) and the distance between the nasion and inion, dividing both measurements by 2 to find the respective mid-points, and marking these with lipliner [see figure 8-7].

Once the Cz is marked, the participant is seated approximately 70 cm away from the computer screen. The EEG cap is placed on the participant's head, ensuring that the Cz (number 16 on the cap) aligns with the mid-point marked by lipliner and the ears are free from hair. The cap is secured with a strap under the chin to provide a comfortable fit and prevent movement during the experiment.

To establish an effective connection between the electrodes and the skin, any hairs obstructing the electrodes are gently moved away using a syringe tip. The objective is to eliminate any barriers that could interfere with the signal. EEG gel is then applied with a syringe to each electrode to get contact with the scalp. An impedance check is then conducted using the g.recorder software. During this process, all electrodes should ideally have an impedance value below 25kΩ before starting the experiment.



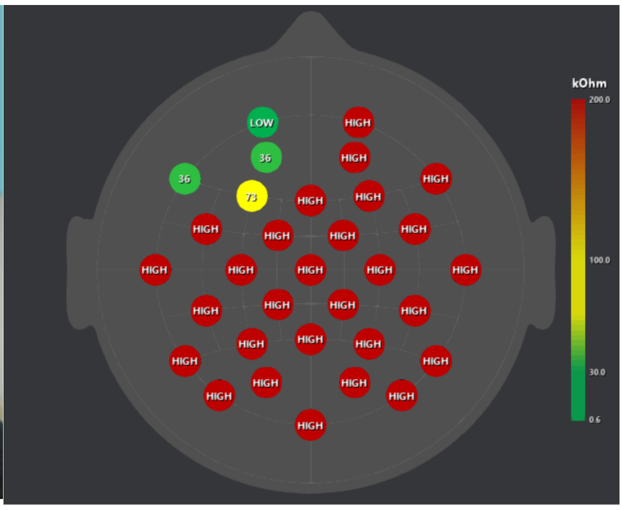


Figure 8-10 (left) Illustrates the process of applying EEG gel to the electrode, captured during the test-run.

Figure 8-11 (right) Displays the screen view while conducting an impedance measurement, as seen in g.recorder.

It is important to be cautious of bridging when applying the EEG gel. Bridging occurs when the gel applied on two nearby electrodes comes in contact with each other, potentially affecting the signal quality.

However, achieving a low impedance for all electrodes can sometimes prove challenging due to factors such as hair thickness. In such cases, the electrodes can be carefully refilled with gel or repositioned using a syringe tip to enhance the signal. A maximum time of 35 minutes is allowed for impedance adjustment. If the impedance remains unsatisfactory after this time, all channels with higher impedance values are noted in the log, and the experiment proceeds.

To ensure accurate eye tracker adjustment, the participant is first asked to lean into the headstand, which is seated approximately 70 cm away from the computer screen. If they wear glasses, the glasses should be pushed up as far as possible to avoid interference. In cases where glasses or lenses cause significant issues, they can be removed, with a note made in the experiment log. The aim is to have both pupils clearly visible on the display.

The eye tracker is then adjusted by turning the red button and dragging the red circles over each eye. Sharpness is fine-tuned using the wheel under the eye tracker, and the left and right arrows are used to display each eye for optimal focus. The participant is asked to look at the four corners of the screen to confirm that the teal-dot part has minimal white, indicating proper alignment.

Pre-calibration of the eye tracker is performed on the host computer by using the auto detect threshold function. This process finds the ideal threshold values for the corneal reflection (CR) and pupil measurements, with the threshold CR ideally ranging between 215 to 240, and the pupil threshold between 60 to 140. The pupil threshold refers to the level of sensitivity at which the ET system detects the pupil, while the corneal reflection threshold pertains to the sensitivity level at which an ET system can accurately detect and differentiate the corneal reflection from other reflections or background noise.

Adjustments to the illumination level may be required to achieve the desired threshold values. The participant's gaze at the four corners of the screen helps to verify if the signal is good, as demonstrated in figure 8-12.

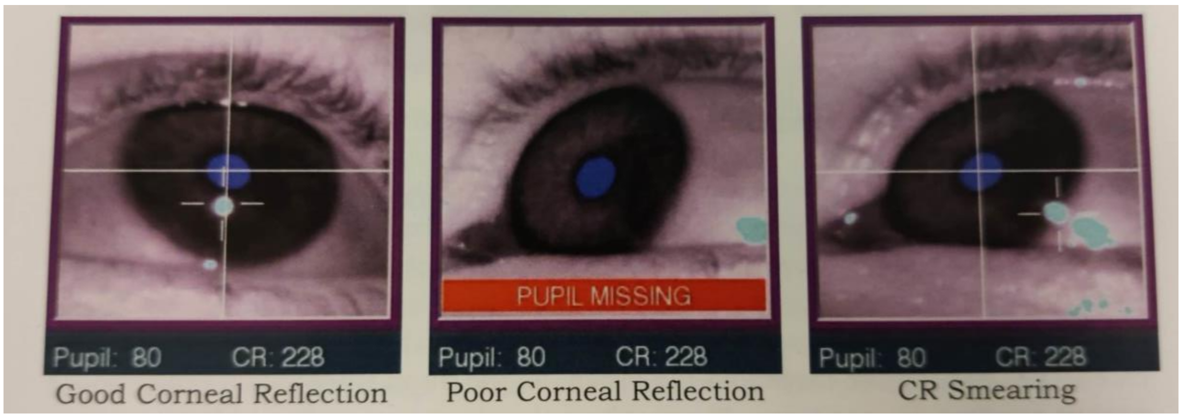


Figure 8-12 This illustration demonstrates the optimal appearance when the participant gazes at the corner of the screen. A deviation of the teal dot from the eye indicates a potentially poor signal robustness, taken from SR Research EyeLink Portable Duo User Manual (SR-research, 2017)

Calibration and validation of the eye tracker require the participant to concentrate on a series of dots that appear on the screen. Throughout this process, the eye tracker maps the location of the participant's pupils in relation to the position of the dots. Nine distinct dots are used, situated in each corner and the center of the screen, to create a comprehensive map of the individual's gaze in relation to the screen. Following the calibration, a validation is conducted to double-check that the measured pupil gaze remains consistent with the calibration data, ensuring accurate ET results.

8.4 Test-run and Improvements made

We recruited a total of 5 participants from our student group. Given that the primary goal of this initial phase was to test the equipment and protocol, we did not place significant emphasis on the demographic composition of the participants. Each participant took part in a single session of each modality. We also trained two of the students to conduct the experiment, this to get feedback on the lab protocol. The experiment was conducted following the step-by-step guide in appendix 1, note that it is the refined and finalized version and may not precisely align with the test-run version.

8.4.1 Hybrid lab setup

Following the test-run, we relocated an ET to the Medtec test lab. Setting up the hybrid fNIRS/ET and EEG/ET systems was relatively simple, as we had already integrated all modalities within the same stimulus software (PsychoPy) and designed the test lab with the ET in mind. Both paradigm programs now initiate the ET setup (calibration/validation) before starting the dot-probe experiment. This hybrid configuration enabled us to shorten the experiment to 2 hours, as participants only needed to complete two dot-probe tasks instead of three, and there was no need for them to visit separate rooms for the experiment.

However, we had to consider two factors when using the hybrid setup. First, the ET can impact the fNIRS and EEG signals. The infrared light emitted from the ET may strike the fNIRS photodiodes, generating substantial noise. We addressed this issue by applying a light-proof shower cap. Second, the ET can produce electrical noise on the EEG signal. While we cannot eliminate this interference, we accept it as a consistent factor across all EEG data collected.

8.4.2 Decreasing fNIRS sampling rate

During the test-run, we encountered an error message indicating cross-talk between channels. The fNIRS system collects data by sequentially activating and deactivating LEDs, a process referred to as the illumination pattern (NIRx, 2018). This is done to ensure accurate measurement, as simultaneous activation of all LEDs would make it difficult for the fNIRS to determine the source of the detected light. At a sampling rate of 3.91 Hz, only one LED is active at a time, as illustrated on the left in Figure 8-13. To increase the sampling rate to 7.82 Hz, two LEDs must be activated simultaneously, as illustrated on the right in Figure 8-13.

LEDs 1-8 are located in the occipital area, while LEDs 9-16 are in the prefrontal area. Given their distance from each other, we did not anticipate cross-talk detection between them. Cross-talk can potentially compromise the data quality. However, this error message appeared for only two of the five participants. Unable to resolve the issue, we opted to decrease the sampling rate to 3.91 Hz. This decision resulted in acquiring half the data, but it ensured the elimination of cross-talk between channels.

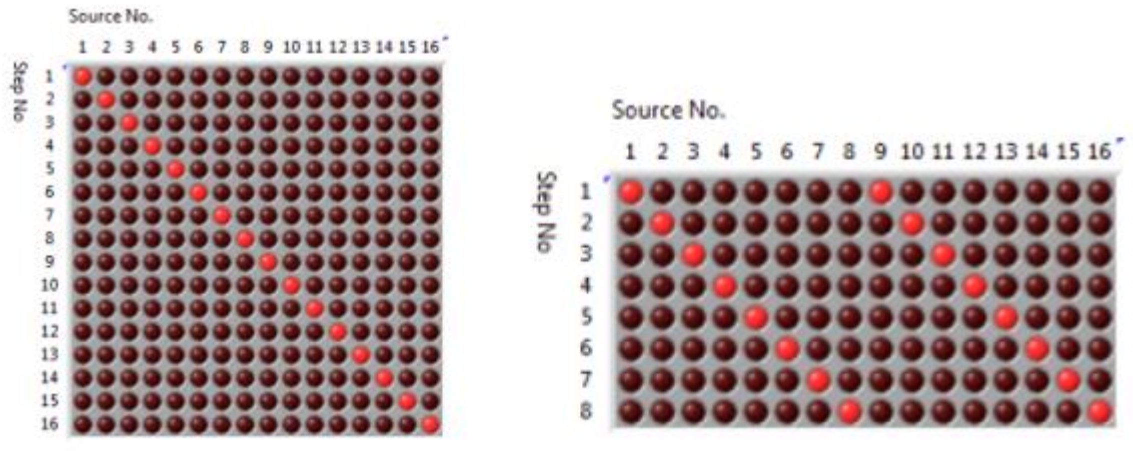


Figure 8-13 Depicts the illumination patterns at different sampling rates; on the left side, a 3.91 Hz sampling rate is shown with only one LED active at any given moment, while on the right side, a 7.81 Hz sampling rate is illustrated with two LEDs simultaneously active, where LED No. 1 is active concurrently with LED No. 9, LED No. 2 with LED No. 10, LED No. 3 with LED No. 12, and so forth. Images are sourced from NIRStar 15-3.

8.4.3 Changing fNIRS paradigm to block design

Upon inspecting the fNIRS data, we observed that our event and rest-time durations were considerably short, with approximately 2 seconds (depending on reaction time) of stimulus and 7 seconds of rest-time. Considering that the hemodynamic response function (HRF) takes about 1-2 seconds to start rising and 5-6 seconds to reach its peak, our measurements essentially captured the decline of the HRF from the previous event at the beginning of the current event. Moreover, (Friston et al., 1994) suggest that the total duration of the HRF impulse is around 26-28 seconds. The low sampling rate of fNIRS further necessitates a longer stimulus and rest time.

Taking these factors into account, we decided to modify the fNIRS paradigm to a block-design, consisting of four trials in each block. Each block will last for approximately 15-16 seconds (dependent on participants' reaction time), followed by a 15-second rest-time. This change results in a total duration of 30 seconds from the presentation of a stimulus to the commencement of the next block. The paradigm will then last approximately 10 min, which is the same as our previous paradigm [See figure 9-4 in results section for illustration of new paradigm].

This alteration is further justified by the fact that the hemodynamic response takes about ten seconds to return to baseline levels after reaching its peak (Nogueira et al., 2022). Moreover, recent research by (Zhang et al., 2021) suggests that a 15-second stimulation duration within an appropriate experimental setup allows researchers to obtain optimal fNIRS signal quality.

8.4.4 Reducing fNIRS trigger setup

We also noticed, when looking at the fNIRS data, that some of the triggers registered were redundant. We used the same trigger setup as the EEG, which sends a trigger e.g., when the fixation cross appears, and when the face stimuli appear. This results in a gap between these two triggers of 1s, which makes sense for the EEG data, since its sampling rate is 1000 Hz. However, for fNIRS with a sampling rate of 7.81 Hz there is almost no data in this gap. Therefore, we changed the trigger setup for the fNIRS so it now only sends a trigger with all the information of the trials at the start of each block.

8.4.5 Integrating ET data with analysis software

We noticed that when looking at the ET data in the EyeLink Data Viewer (ET analysis software), our stimulus wasn't showing up, and the data wasn't divided into separate trials. Our variables also weren't visible in the software. Although analysis og the data could be done using other software like Python or MATLAB, we wanted to use EyeLink Data Viewer to make the analysis process as easy as possible for future researchers.

To do this, we first needed to split the dataset into individual trials. We sent a specific input at the beginning of each trial to let the software know when a new trial started: el_tracker.sendMessage('TRIALID %d' % trial_index).

Then, we added the variables to the software by sending another input for each variable, such as the one for face pairs: el_tracker.sendMessage('!V TRIAL_VAR Face_Pairs %s' % Face_Pairs). We did this for all variables, including reaction time and accuracy, at the end of each trial.

Next, we wanted to show the images from the experiment in the Data Viewer. We figured out the size and location of each image and sent the input with the necessary information: el_tracker.sendMessage('!V IMGLOAD CENTER %s %d %d %d %d' % (image_face_left, left_image_center_x_axis, image_center_y_axis, image_width, image_height)).

We also needed to define "areas of interest" – in our case, the face images – to analyze when participants looked at each image and for how long. We did this by sending an input with the appropriate information: el_tracker.sendMessage('!V IAREA RECTANGLE %d %d %d %d %d %s' % ia_image_left).

However, we faced a problem with message errors because the software received multiple messages at the same time. To fix this, we added a counter to control the message sending rate and prevent them from being sent simultaneously.

8.4.6 Figure size and location

In our experiment, the picture sizes and locations for our paradigm were not correctly set up, causing the pictures to be too large and too close to each other. To address this issue, we adopted the sizes and locations used in the dot-probe paradigm from the study by (Andrzejewski & Carlson, 2020). They used facial stimuli subtending 5° × 7° of the visual angle, with the innermost borders of the two facial stimuli separated by approximately 14° of the visual angle.

To calculate the appropriate sizes, we first determined the distance from the screen at which participants would sit, taking into account the requirements of the ET. We used SR-Research's Trackable Range Calculator (SR-Research, 2023) to determine the optimal distance, ensuring that the ET could effectively track the participant's gaze when looking at the corners of the screen. For our 53 cm wide and 30 cm tall monitor, the recommended distance was 115 cm.

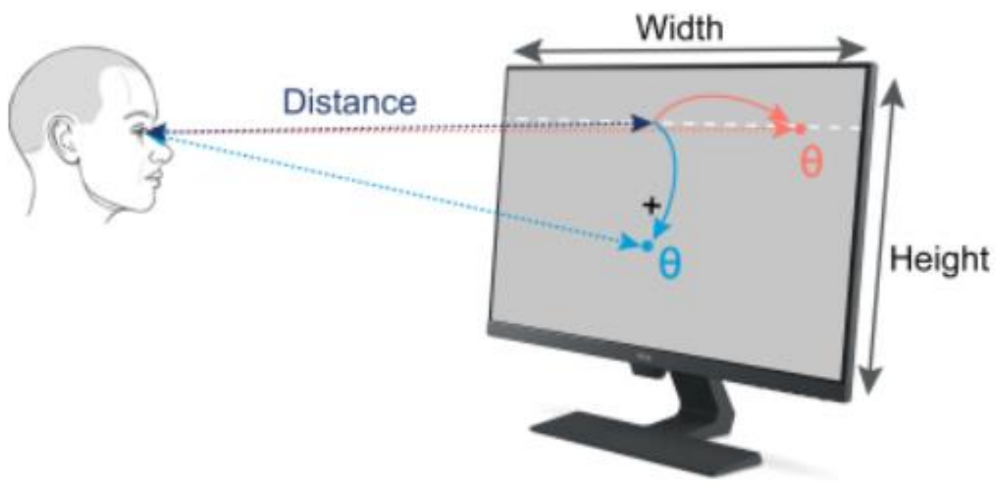


Figure 8-14 Visual representation of the angular measurements displayed on the screen, adapted from (SR-Research, 2023).

PsychoPy utilizes pixels for specifying the location and size of images, necessitating the conversion of the dimensions to pixels. Our screen has a resolution of 1080p (1920x1080), and by dividing the width of the screen in pixels by the width in centimeters, we determined that there are 36.2 pixels per centimeter (1080 / 53 = 36.2 pixels per cm).

We then calculated the dimensions of the images on the screen using the following formula: w = arctan(α) * d, where w represents the width of the picture on the screen, α is the visual angle, and d is the distance from the eyes to the screen (115 cm in our case).

Calculating the height of the images in centimeters (h = arctan(7) * 115), we arrived at 13.98 cm. Converting this to pixels, we obtained a height of approximately 506 pixels (h_pixel = 36.2 * 13.98 ≈ 506 pixels). Similarly, we calculated the width of the images as 10.01 cm (w = arctan(5) * 115) and converted it to pixels, yielding a width of approximately 362 pixels (w_pixel = 36.2 * 10.01 ≈ 362 pixels).

To determine the distance from the center of the screen to the innermost border of the picture (left border for the right picture and vice versa), we accounted for PsychoPy's center-to-center measurement by adding half of the width in pixels to the distance. With the picture being 7° from the center to the nearest picture border, we knew that 7° corresponded to 506 pixels. Therefore, the pictures should be placed 687 pixels from the center (506 + 362/2 = 687 pixels).

The participant will now sit 115cm from the screen, instead of 70cm.

8.4.7 Making the experiment more engaging

Upon the completion of the experiment, we got feedback from the participants. They expressed that the test was monotonous, causing them to lose interest. They also mentioned that they could nearly predict the pattern of the program due to consistent durations. Furthermore, they suggested that the task's introduction could be more comprehensible.

To address the issue of boredom, we drew inspiration from (Carlson & Fang, 2020). In their study, participants received feedback on their reaction times after each block to encourage accurate and rapid responses. Consequently, we incorporated a display that presents the participant's fastest reaction time achieved so far at the end of each trial. For fNIRS/ET, this occurs after each block, while for EEG/ET, it appears at the end of each trial. To allow participants sufficient time to read their reaction times, we extended the pause between trials from 1 second to 1.5 seconds. In order to maintain consistency between the hybrid fNIRS/ET and EEG/ET paradigms, we also included this pause in the fNIRS/ET paradigm.

To eliminate the predictable "rhythm" of the paradigm, we took cues from (Kappenman et al., 2015), who introduced a time jitter intertrial interval of 750–1250 ms during the fixation cross section of the dot-probe task. This approach not only makes the paradigm more engaging by introducing unpredictability, but also helps to pseudorandomized the length of the inter-trial interval. This, in turn, avoids any temporary physiological correlations with the stimulus, such as breathing patterns, thus strengthening the experiment's robustness (Meryem A. Yücel, 2021).

Lastly, we provided a straightforward explanation of the dot-probe task before the test commenced, allowing participants to initiate the task by pressing a button once they fully understood the instructions and were prepared to begin.

8.4.8 Stimuli duration

Upon examining the ET data, we observed that participants' eye movements were minimal. Factors such as the size and spacing of the face stimuli pictures probably contributed to this outcome. However, further review of the relevant literature indicated that our face stimulus duration of 200ms might have been too brief (Price et al., 2015; Stevens et al., 2011). Studies have demonstrated better results with longer durations such as 600ms (Stevens et al., 2011) when using electrooculogram, and 2000ms (Price et al., 2015). By increasing the presentation duration to 1200ms, we aim to enhance the accuracy and reliability of our study, while keeping the overall length of the paradigm manageable.

8.4.9 Tiredness VAS-scale

Several participants reported experiencing fatigue prior to the commencement of the test. In order to monitor and control for tiredness, a Visual Analog Scale (VAS) score will be incorporated at the start each of the experiments. The VAS scale is employed to gauge the participants' level of tiredness by posing the question, "How tired are you?" Participants respond using the scale, which ranges from 0 to 9, with 9 representing the highest level of fatigue and 0 signifying complete alertness. This approach ensures a comprehensive understanding of the participants' fatigue levels throughout the testing process, allowing for more accurate data interpretation.

8.4.10 Response-box

We have substituted the traditional keyboard with a high-precision button box, which comes bundled with the ET. This response device boasts millisecond accuracy and features four buttons. Minimizing delay is crucial for obtaining reliable data from the EEG and ET systems, as they possess exceptionally high sampling rates.

To enhance user experience and reduce confusion, we have labelled one button with an 'L' for 'left' and another with an 'R' for 'right.' This ensures that participants can easily identify the appropriate buttons to press during the study.

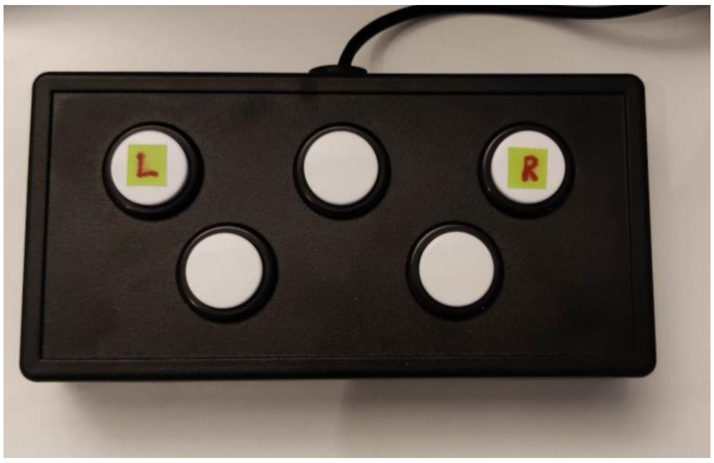


Figure 8-15 This image showcases the response box, as utilized in our study. The photograph has been captured from the MedTec test Lab.

8.5 CONDUCTING EXPERIMENT

Having completed the laboratory setup, experimental paradigm, and experimental protocol, we were fully prepared to commence data collection. Before initiating this process, we trained seven undergraduate and graduate students from OsloMet

University to assist in conducting the experiment. A comprehensive description of the experimental paradigm employed will be provided in the subsequent Results and Discussion sections.

8.5.1 Ethics and storing of data

Collecting experimental data from participants requires careful consideration of ethical issues to ensure that their rights, well-being, and privacy are protected. Informed consent, which involves explaining the nature and purpose of the study, potential risks, and benefits, as well as participants' right to withdraw, is crucial. Researchers should obtain written consent before starting the study. To minimize any potential harm or discomfort, the study should be designed and conducted in a way that does not cause undue physical or emotional stress. Confidentiality of personal information and research data is critical and should be maintained using secure storage and access controls.

Before starting the study, ethical approval from an institutional review board or ethics committee is necessary. In our case, we applied for ethical approval from the Regional Committee for Medical and Health Research Ethics (REK) and stated that only "healthy" participants would be included. To protect privacy, each participant was assigned a unique ID number, and personal information was stored in a separate paper document under lock and key. All data will be initially stored on a local device and then uploaded to a cloud service called "Tjenester for sensitive data" (TSD), designed specifically for sensitive data storage. Consent forms will also be stored separately and deleted from the researcher's email inbox.

8.5.2 Participants and recruitment process

Ensuring that the demographic of the participants is as similar as possible is crucial in conducting a psychology experiment. In this project we have decided to recruit 60 healthy adult females between the ages of 18 to 30 years. A higher participant value of 60 is beneficial because it increases the statistical power of the study, making it easier to detect meaningful effects.

To ensure the participants are representative of the general population, we have established some exclusion criteria, including medication use, severe psychopathology, brain injury, and neurological disorder. These exclusion criteria were put in place to minimize the potential confounding variables that could affect the results of the study.

To recruit participants for the experiment, we used multiple methods such as personal networks, social media, and Instagram ads. Participants were directed to an online survey portal called "Nettskjema" by scanning a QR code. This service, provided by the University of Oslo (UiO), allowed for secure data storage with restricted access limited to researchers and students related to the project. Upon meeting eligibility criteria, participants were contacted by the researcher to schedule an appropriate time for testing. A consent form was sent to the participants, and they were asked to return a signed form before the scheduled experiment.

8.5.3 Procedure

Participants were instructed to convene with the researcher in the lobby of Building P35 at OsloMet University. Upon arrival, they were escorted to the medtech test laboratory. Here, they first engaged in the hybrid fNIRS/ET experiment, followed by the hybrid EEG/ET experiment. The procedures outlined in Appendix 1's step-by-step guide were strictly adhered to throughout the process. Upon completion, each participant received a gift card valued at 500 NOK as a token of appreciation.

8.6 PRE-PROCESSING AND ANALYSIS OF FNIRS DATA

For fNIRS data pre-processing and analysis, we employ the **Satori software**, developed collaboratively by Brain Innovation and NIRx. Satori was designed with a strong emphasis on user-friendliness, facilitating effortless analysis of fNIRS data. The software supports the standardized near-infrared file format (sNIRF) and is compatible with earlier NIRScout header-based formats. Moreover, Satori's flexible interfaces enable third-party applications to access processed data and computed statistics, but since our goal was to make an easy to use pipeline for pre-processing and analysis, we opted to only use the methods available in Satori.

Satori's Workflow Manager option allows us to pre-process and analyse multiple fNIRS datasets using identical parameters. The order of steps can be adjusted freely within the manager, which is divided into two sections—the workspace and the toolbox. Workflow items can be effortlessly dragged and dropped into the workspace and connected in a specific sequence. Available items encompass Input/Output, Transformations, Preprocessing, Postprocessing, Analysis, and Tools. The Event Manager item permits alterations in condition durations within the protocol, while the Trim Data item enables the removal of undesired data.

fNIRS signals are influenced by biological factors such as respiration and movement, which can introduce noise and artifacts, including motion artifacts that cause sudden spikes and baseline shifts (data abruptly jumps to a new value and remains there). To address these issues, pre-processing is necessary to eliminate artifacts, correct signal drift, and normalize the data. Detaild explanation of the pre-processing is presented in the Result and Discussion section.

After pre-processing the data, we conducted an analysis using a multi-subject General Linear Model (GLM) approach, where we followed the Satori Multi-Subject GLM Guide (Lührs et al., 2022) and Satori user manual (Brain-Inovation, 2023). We have used these guides to give an simple explanation of the logic behind the GLM analysis in Satori:

fNIRS measures brain activity by looking at changes in blood oxygenation levels in specific regions of the brain. The GLM aims to understand how these changes are related to different experimental conditions.

In simple terms, the GLM tries to explain the observed fNIRS time course (dependent variable) as a combination of several reference functions (independent variables). These reference functions represent the expected fNIRS responses for different experimental conditions. They are sometimes called predictors, regressors, explanatory variables, covariates, or basis functions.

The GLM uses a design matrix, which is a table that contains the reference functions (predictors). Each predictor is associated with a coefficient or beta weight, which quantifies its contribution in explaining the observed fNIRS time course. The model also accounts for error values, which represent the difference between the actual data and the predicted data.

To summarize, the GLM tries to find the best combination of predictor time courses and their corresponding beta weights to explain the observed fNIRS time course. The resulting beta weights can provide insights into the brain's response to different experimental conditions.

Here's a simplified explanation of the GLM equation:

$$y = b0 + b1 * X1 + b2 * X2 + .. + bp * Xp + e$$

y: the observed fNIRS time course (dependent variable)

b0, b1, b2, ... bp: beta weights (coefficients) for each predictor

X1, X2, ... Xp: predictor time courses (independent variables)

e: error values (difference between the actual data and predicted data)

The goal of the GLM is to estimate the best beta weights for each predictor so that the model can accurately explain the observed fNIRS time course. Once the beta weights are determined, they can be used to interpret the brain's response to different experimental conditions. A large positive beta weight indicates strong activation during the condition, while a large negative beta weight indicates strong deactivation.

Correction for serial correlations is a crucial step in the analysis of fNIRS data using the General Linear Model (GLM). The GLM assumes that the residuals (the noise in the data) are uncorrelated. However, fNIRS data often contain serial correlations due to trends or physiological noise. To improve the accuracy of the GLM, these correlations need to be removed. Satori uses a process called pre-whitening to remove these, more info about this can be found in the Satori user manual (Brain-Inovation, 2023).

We selected Separate Subject Analysis, which involves estimating subject-specific beta values for each subject and condition within the Multi-Study GLM list. This approach follows the classical methodology for calculating a Random Effects GLM, which allows for potential generalization of effects beyond the measured sample.

9 RESULTS AND DISCUSSION

9.1 THE FINISHED LAB SETUP AND PROTOCOL

We established a state-of-the-art test lab, accompanied by a detailed lab protocol (refer to Appendix 1). After conducting an initial test-run, we were able to identify and implement several significant improvements. These enhancements have led to the development of a highly efficient, user-friendly test lab environment. We have collected valuable data from a total of 60 participants using this optimized setup.

The test lab is fully equipped with a single PC containing all the necessary software to operate the three modalities and the stimulus software. Additionally, all the required hardware components are connected to this central PC. This streamlined configuration allows for seamless integration and ease of use, making it an ideal platform for future researchers to conduct further experiments.

The images below provide a visual representation of the lab setup:

The first image provides an overview of the test lab, illustrating two distinct sections. On the left side, we see the researcher's workstation and Host-PC for ET, while the right side displays the seating arrangement for the participant. These sections are separated by a small wall. Throughout the experiment, a researcher would occupy the left side, closely monitoring the signal graphs to ensure a smooth process without any complications.

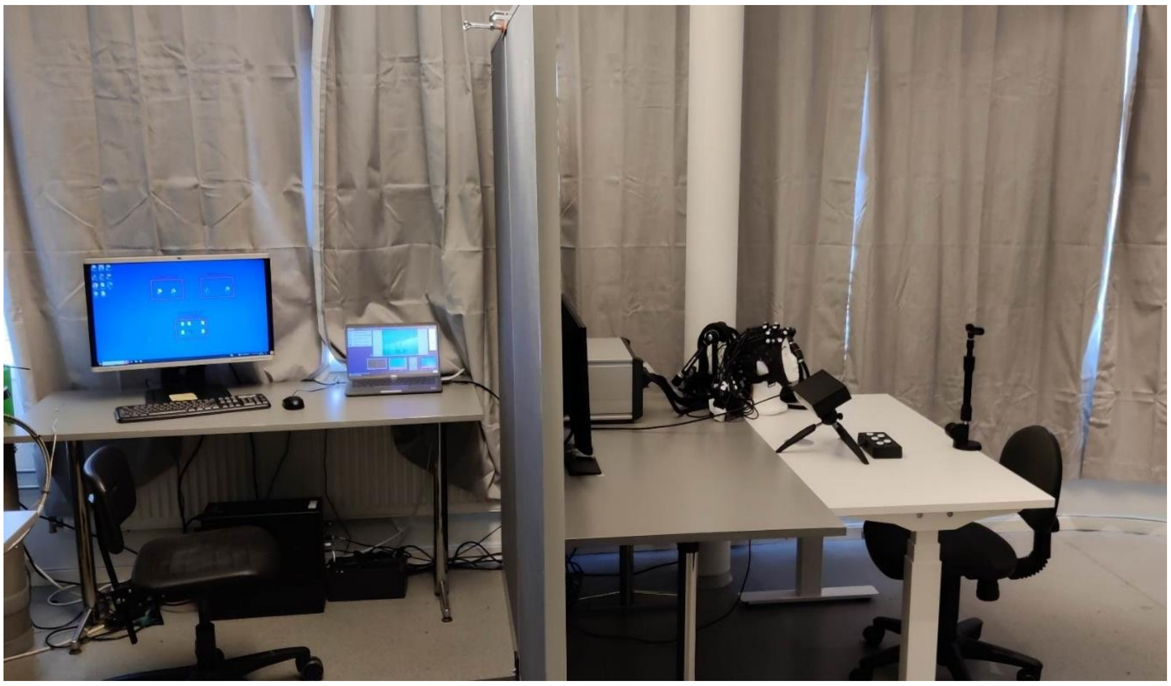


Figure 9-1 Overview of the test lab.

The second image illustrates the participant's viewpoint during the experiment. It offers a comprehensive perspective of the three modalities: the EEG system positioned to the left of the screen, the fNIRS setup with a mechanical arm to the right, and the ET placed directly in front of the screen. On the left side of the image, the step-by-step lab protocol is prominently displayed. Meanwhile, the response box and headstand are conveniently positioned at the center.

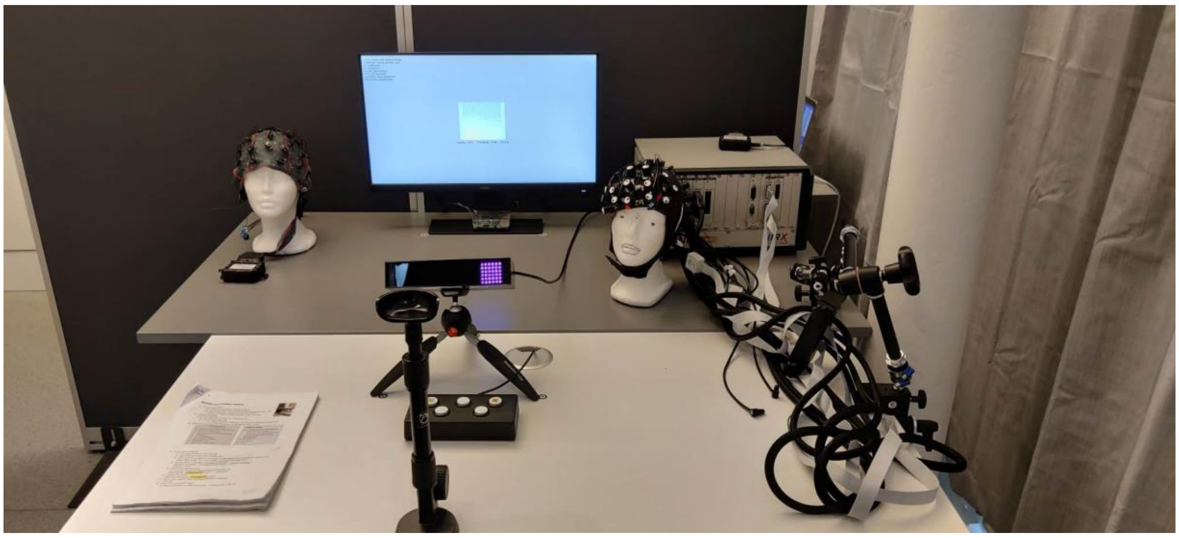


Figure 9-2 Showcase of the test lab, illustrating the participant's seating arrangement during testing.

In the third image, we observe a participant who is resting their head on a headstand while wearing a securely mounted fNIRS cap. This setup indicates that the participant is ready for the calibration process of the fNIRS system.

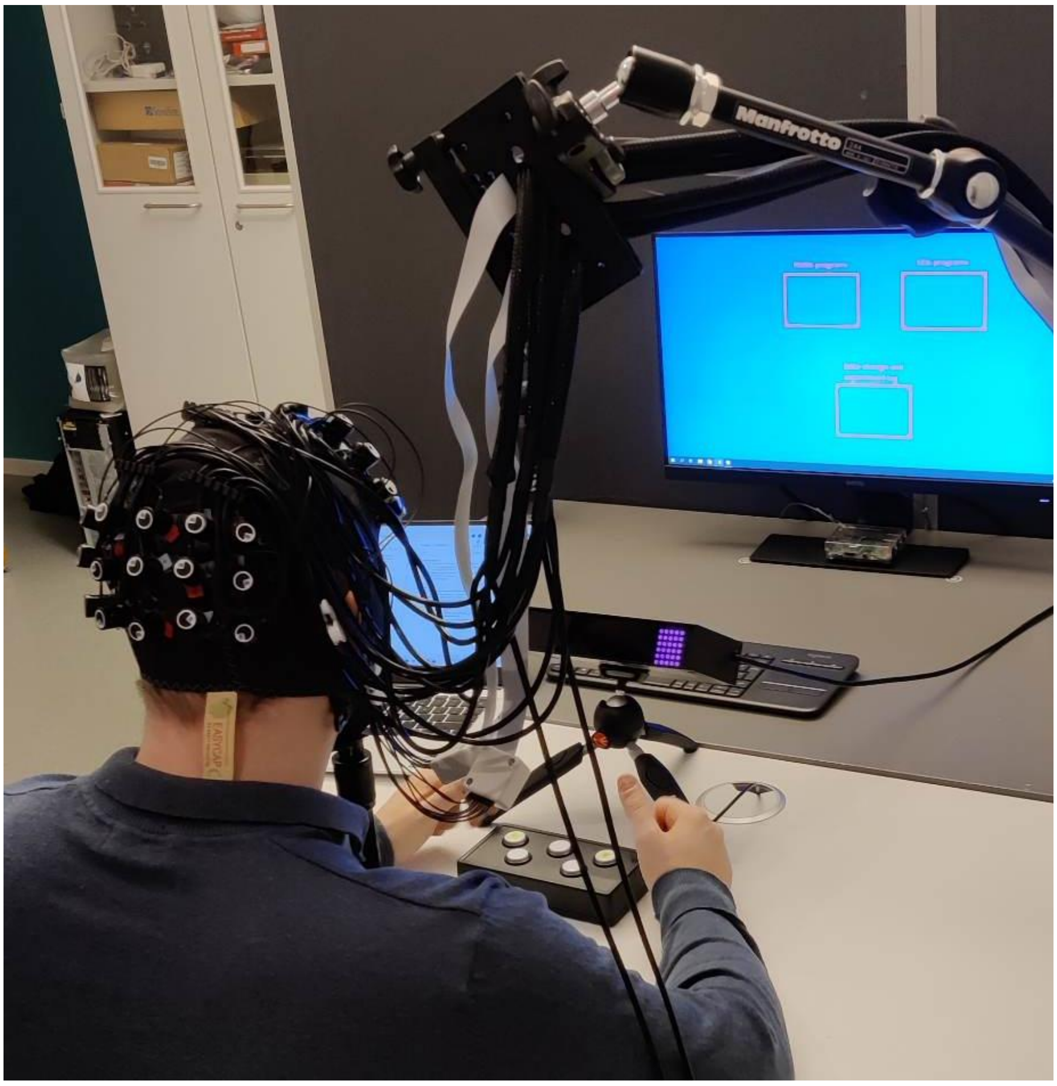


Figure 9-3 Showcase a person during the pre-calibration of fNIRS.

9.2 THE FINISHED EXPERIMENT PARADIGM DESIGN

We have developed two dot-probe task paradigms in Psychopy: one integrated with the hybrid fNIRS/ET system and the other integrated with the hybrid EEG/ET system. Both paradigms were designed to be as similar as possible, enabling a comparison of data between the two hybrid modalities. However, they have been fine-tuned to accommodate the specific requirements and characteristics of each modality.

Here is an explanation of the two paradigms, starting with what is common between the two:

At the beginning of the paradigm, the ET pre-calibration is automatically initiated. Once the ET calibration is completed, a VAS for tiredness appears, prompting participants to rate their level of tiredness on a scale of 0 to 9 (with 9 indicating the highest level of tiredness). Following the completion of the tiredness rating, the experiment is ready to commence.

The experiment starts by providing participants with information about the dot-probe task and instructing them to press any key when they are prepared to begin. A 30-second countdown timer appears on the screen before the experiment initiates.

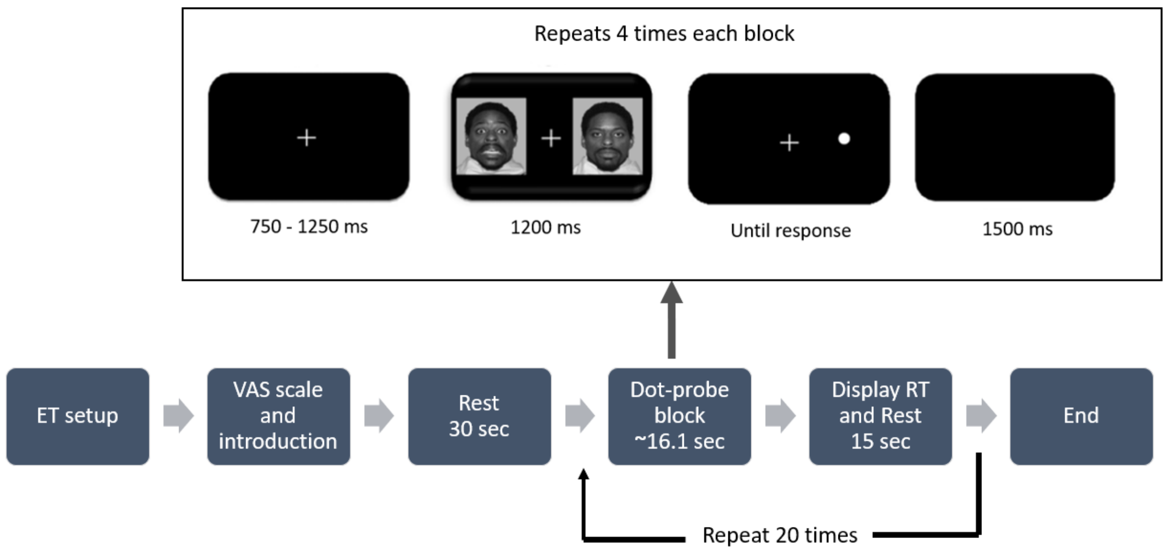
Each trial starts with a white fixation cross displayed at the center of the screen for a randomized duration of either 750ms or 1250ms. The fixation cross remains visible until the dot disappears. Two faces (happy/neutral, fearful/neutral, or neutral/neutral pairs) are presented for 1200ms, spanning 5° × 7° of the visual field, with roughly 14° separating the innermost borders of the facial stimuli. Immediately after the faces vanish, a white dot appears behind either the left or right image, centered within the picture. Trials are categorized as congruent if the dot is behind the face with the most negative emotion (e.g., behind the neutral face in happy/neutral pairs), and incongruent otherwise.

Participants are required to indicate the dot's location by pressing the 'left' or 'right' button on the response box with their index finger. Following their response, a black screen is displayed for 1500ms, marking the end of the trial.

For fNIRS/ET paradigm the participants complete twenty blocks, each consisting of four trials. Each condition is presented an equal number of times (four instances per condition): neutral/neutral, happy/neutral congruent, happy/neutral incongruent, fearful/neutral congruent, and fearful/neutral incongruent. Each block maintains an equal ratio of male to female faces, dot locations (left/right), and time jitter (750ms/1250ms). Additionally, a unique face is displayed in every trial, ensuring no repetitions. The participant goes through a total of 80 different trials.

After completing each block, participants are shown their best reaction time, followed by a 15-second rest period. The order of blocks and trials within them is counterbalanced across participants.

The complete code for the fNIRS/ET paradigm is available in Appendix 3.



Repeats 4 times each block
750 - 1250 ms
1200 ms
Until response
1500 ms
ET setup
VAS scale and introduction
Rest 30 sec
Dot-probe block ~16.1 sec
Display RT and Rest 15 sec
End
Repeat 20 times

Figure 9-4 Illustration of the hybrid fNIRS/ET paradigm design.

For the EEG/ET paradigm it's the same procedure, but without the blocks and 15sec rest periods. Participant is presented with the exact same conditions and the same ratio of male to female faces, dot locations (left/right), and time jitter (750ms/1250ms). The participant completed a total of 160 trials, where all of them where psudoranomised. The participants RT was displayed after each trial.

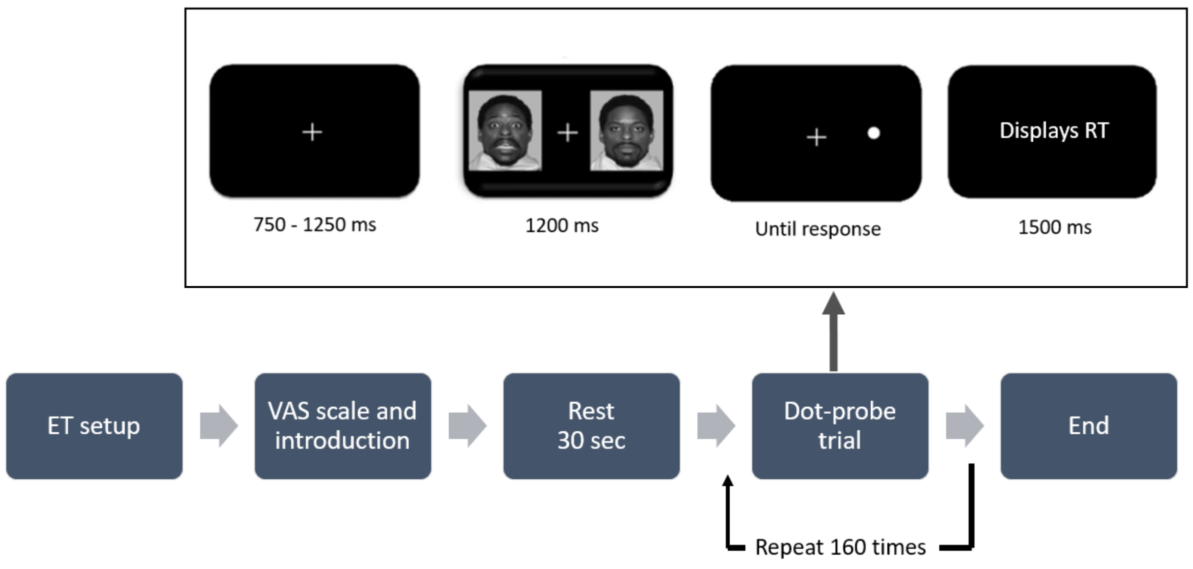


Figure 9-5 Illustration of the hybrid EEG/ET paradigm design.

The complete code for the fNIRS/ET paradigm is available in Appendix 4. It is worth mentioning that the code includes additional components such as a resting-state task and a visual search task. These were not elaborated upon in the explanation as they were part of a separate project undertaken by other researchers and do not contribute to the primary objective of the master thesis.

Here is a showcase of the new trigger setup for the modalitites:

Stimulus	**FNIRS**
Experiment Start	1
Neutral/Neutral Block	2
Happy/Neutral Congurent Block	3
Happy/Neutral Incongruent Block	4
Fearful/Neutral Congurent Block	5
Fearful/Neutral Incongruent Block	6
End of Block	7
Experiment Ended	8

Table 9-1 Overview of the new and improved fNIRS trigger setup.

Stimulus	**EEG**	**ET (See section 8.4.5 for more in depth about integration with analysis software)**

Experiment Start/Stop	1	‘Experiemtn_Start’ / ‘_Stop’
Fixation Cross	2	‘Fixation_Cross_Start’ / ‘_Stop’
Face Pairs: Neutral/Neutral	4	‘Face_Pairs_Neutral_Neutral_Start’ / ‘_Stop’
Face Pairs: Happy/Neutral	8	‘Face_Pairs_Happy_Neutral_Start’/ ‘_Stop’
Face Pairs: Fearful/Neutral	16	‘Face_Pairs_Fearful_Neutral_Start’ / ‘_Stop’
Dot Congruent	32	‘Dot_Congruent_Start’ / ‘_Stop’
Dot Incongruent	64	‘Dot_Incongruent_Start’ / ‘_Stop’
Reaction (keyboard pushed)	128	‘Reaction’

Table 9-2 Overview of the EEG and ET trigger setup.

9.3 FNIRS PRE-PROCESSING PIPELINE

A standardized workflow has been developed, and all datasets have been processed using the same procedure. The sequence of pre-processing steps follows the recommendations outlined in the Satori user manual (Brain-Inovation, 2023) and the guide made by (Pinti et al., 2019). Each stage of the pre-processing process will be detailed in this section, adhering to the workflow order. We implemented two distinct workflows for this purpose.

The first workflow included removal of masked channels, trimming, channel rejection, conversion, and event editing. Once completed, all data was saved in a folder named "data_ready_for_pre_processing." Subsequently, we renamed each file to include only the participant number. This was done based on Satori's recommendation to remove the date and time from filenames, as it could cause confusion during analysis by making the software perceive two different subjects as the same (Lührs et al., 2022). Moreover, this simplified the process of experimenting with various pre-processing techniques.

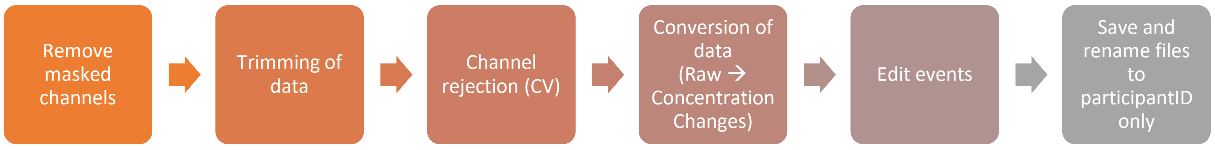

Remove masked channels
Trimming of data
Channel rejection (CV)
Conversion of data (Raw → Concentration Changes)
Edit events
Save and rename files to participantID only

Figure 9-6 Illustration of workflow one (Methods Employed Enclosed in Parentheses).

The second workflow contained motion artifact removal, temporal filtering, and normalization.

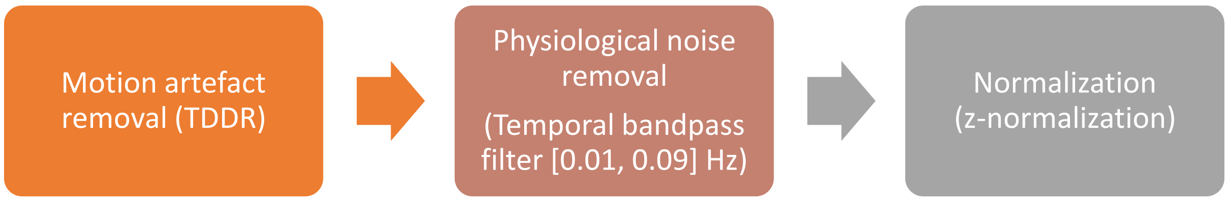

Motion artefact removal (TDDR)
Physiological noise removal
(Temporal bandpass filter [0.01, 0.09] Hz)
Normalization (z-normalization)

Figure 9-7 Illustration of workflow two (Methods Employed Enclosed in Parentheses).

9.3.1 Remove masked channels

The fNIRS data file reports that it comprises 256 channels. Nonetheless, only 42 of those channels are relevant to our analysis, as the remaining channels do not contain any data. Fortunately, our analysis software, Satori, automatically masks channels without data.

During the visualization stage, both 2D and 3D representations display all 256 channels, including the masked channels. This approach tends to complicate the visualization and hinder interpretation. To enhance the analysis and visualization, we removed all masked channels at the beginning of the pre-processing phase. This method enables us to solely analyse and visualize the 42 channels of interest and produce a clearer and more interpretable outcome.

9.3.2 Trimming of data

Trimming data is a necessary step in improving the quality of subsequent analyses. To ensure that we only analyse relevant data, we use a standardized approach that involves sending a trigger 1 at the beginning and a trigger 8 at the end of each experiment. This approach enables us to identify the actual experiment and remove all data that is not relevant to it.

Consequently, we remove all data before trigger 1 and after trigger 8 to eliminate any unnecessary data. This step is crucial as unnecessary data can lead to the removal of good channels and false results. By removing all unnecessary data, we can ensure that our analyses are based on high-quality data that accurately reflects the actual experiment.

9.3.3 Channel rejection

The quality of data acquired in fNIRS experiments can sometimes be inadequate, leading to the need for removal of one or more channels from further analysis. This is achieved through the channel rejection option, which offers two criteria for identifying channels that require rejection: the **Coefficient of Variation (CV)** and the **Scalp Coupling Index (SCI).**

CV is a measure of the variation in the signal, calculated as the percentage ratio of the standard deviation to the mean of the raw data. A threshold value is set, typically 7% for conservative estimates, while a threshold of 10% or higher is used for more liberal estimates. Only the raw data is used to calculate the CV. Here is the formula:

$$CV(\%) = 100 \times \frac{std(data)}{mean(data)}$$

SCI involves filtering the signal to preserve only the heartbeat band and subsequently filtering the data in the frequency range of 0.5-2.5 Hz before calculating the correlation between the optical density (OD) wavelengths. A correlation coefficient below 0.75 is deemed too low and warrants channel rejection.

If the sampling rate is adequately high, such as 10 Hz, the heartbeat can serve as a reliable indicator of the coupling between the optode and the scalp. Therefore, SCI can be used as a quality control metric for the fNIRS signal (Meryem A. Yücel, 2021). In our case we have a sampling rate of 3.9Hz, which is low, therefore we use CV as our method of rejecting channels.

When determining the maximum acceptable coefficient of variation (CVmax%) for an fNIRS experiment, it is essential to consider the type of test being performed. In tests where the participant is highly active, such as those involving physical exercise or movements, the contact between the detector and the scalp is more susceptible to change, leading to more noise and signal distortion (Piper et al., 2014). If a CVmax% value that is too high is accepted, the likelihood of noise and distorted signals increases. Conversely, if the value is too low, valuable information may be excluded by rejecting channels unnecessarily.

In a study involving infants, a CVmax% of 10% was used (Blasi et al., 2014), while another study involving physical activity (cycling) used a CVmax% of 15% (Seidel et al., 2019). In both cases, it was assumed that the participants were highly active.

In our study, participants move their fingers, but are otherwise still with their head resting on a headstand. Based on these examples and our own experiment, we have selected a CVmax% value of 7%.

To ensure that our chosen technique met the required standards and that our analysis did not include poor quality channels, we performed the following validation steps:

- We inspected the 10 worst-performing channels based on the channel rejection method.

- Upon examination, all of the "Bad" (red) channels were rejected, along with many of the "Acceptable" (yellow) channels.

Based on these findings, we concluded that manual channel rejection was not necessary. The automatic CV channel rejection technique effectively eliminated the bad channels and retained only the good quality data for our analysis.

9.3.4 Conversion of data

It is essential to transform raw data into valuable information about oxygenated and deoxygenated hemoglobin levels (HbO and HbR). This is done by analyzing light intensity changes from the emitter and detector positions. Doing so helps us interpret fNIRS signals and comprehend the neural activities being investigated. The steps involved include converting intensity time-series into attenuation shifts (optical density) and then into concentration changes of HbO and HbR. Although Satori doesn't reveal its calculation method, it's typically done using the modified Beer-Lambert law (Delpy et al., 1988).

9.3.5 Edit events

In the edit event section, users can modify the triggers within their data. To clarify the distinction between triggers and events, a trigger refers to the signal sent during data collection that indicates when an important stimulus is presented, while an event specifies the duration of each of these triggers. For simplicity, we will refer to both as "events" in the following sections.

Initially, the first and last events are removed, as these simply indicate the start and end of the experiment and are not necessary for the analysis. Their primary purpose is to aid in data trimming.

Next, we specify the duration of each event. However, there is a challenge in this step. Since reaction times vary for each trial, the events in the experiment have different durations. Satori, unfortunately, only allows setting all events to the same duration. Manually adjusting the duration for each event is time-consuming, as it involves going through 20 different events for 60 participants.

To overcome this issue, we calculated the mean duration of each event by finding the mean reaction time of all 60 participants and then used that to determine the duration of each event. The mean event time was found to be 16.1 seconds, with a standard deviation of 0.37 seconds.

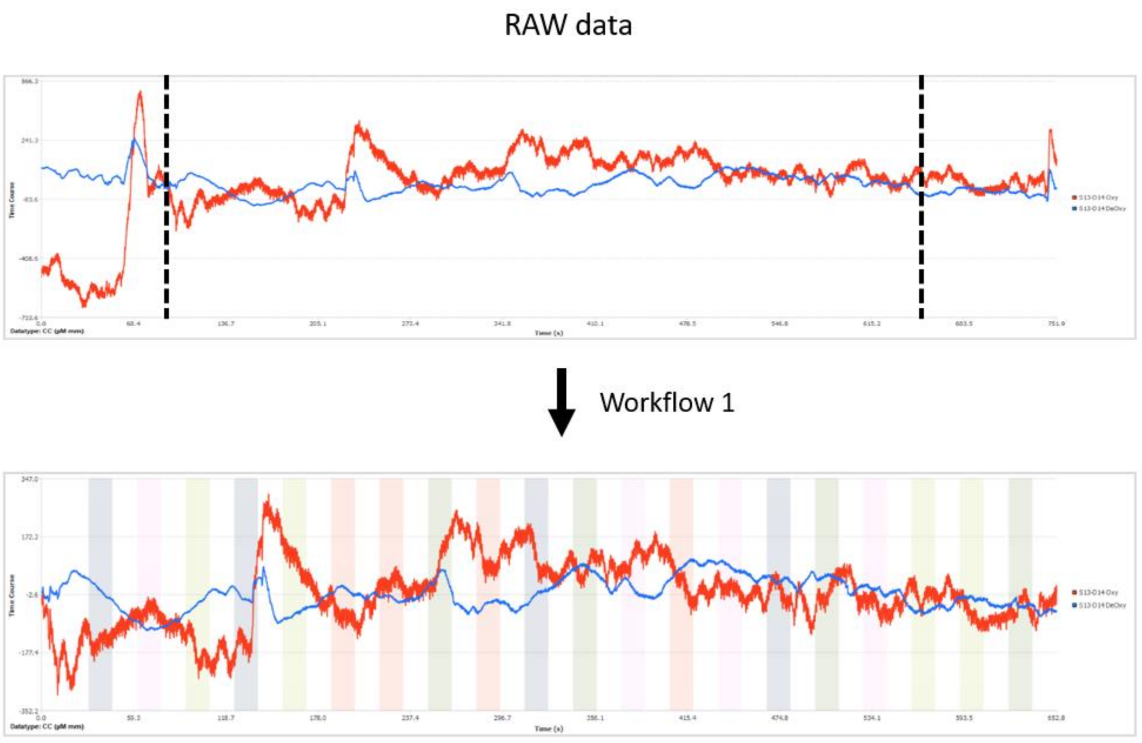


Figure 9-8 Illustration of data processing in Workflow 1, showcasing the significant trimming of prominent motion artifacts at the beginning and end of the raw data (dashed lines). The data is now segmented into distinct events, represented by colored blocks corresponding to five different conditions (happy/neutral congruent, neutral/neutral, etc.). Image made in Satori and edited in PowerPoint.

This discrepancy may potentially affect the fNIRS analysis results as some blocks could inadvertently include data from the rest periods, while others might exclude valuable data. Despite this issue, there are several reasons to believe that the impact on the overall analysis is minimal. First, the fNIRS technology inherently has a low sampling rate, which means that the number of data points collected within the short time windod. Consequently, the potential loss or inclusion of data points due to the fixed 16.1-second block duration is relatively small.

Second, the variation in the actual duration of each block is minimal, as the standard deviation is only 0.37 seconds. This small deviation means that any discrepancies introduced by the fixed block duration are unlikely to significantly distort the results of the fNIRS analysis.

9.3.6 Motion Correction

Head movements can introduce noise into recorded data, particularly affecting neuroimaging techniques. Although fNIRS signals are more resistant to motion artifacts compared to fMRI and EEG/MEG methods, they are still susceptible to disturbances caused by head and skin movements (Pinti et al., 2019).

Motion artifacts display various shapes, frequency content, and timing, ranging from easily detectable high-amplitude, high-frequency spikes to low-frequency content that is difficult to distinguish from normal HRF. These artifacts can be classified into three groups: spikes, baseline shifts, and low-frequency variations (Pinti et al., 2019). Baseline shifts may occur when the optode settles on a different location after motion, while slow head movements can generate low-frequency artifacts (Jahani et al., 2018).

These artifacts may be isolated events or temporally correlated with the HRF. Consequently, the effectiveness of motion artifact correction techniques varies depending on the type of artifact, making the optimal approach data-dependent (Brigadoi et al., 2014). Due to the broad range of frequencies encompassed by motion artifacts, correcting them solely through frequency filtering is challenging without affecting HRF estimation (Jahani et al., 2018).

In the process of addressing motion artifacts, it is essential to balance the need for artifact removal with the risk of inadvertently eliminating valuable data. In our experiment, participants were instructed to move only their index fingers and sit as still as possible. To further minimize head movement, they rested their heads on a headstand. As a result, we expected minimal motion artifacts in our data, except for those caused by eyebrow movement.

Satori offers four methods for addressing MA: the spike removal algorithm, manual spike removal, the Temporal Derivative Distribution Repair (TDDR), and the Correlation Based Signal Improvement (CBSI). Initially, we tried the spike correction algorithm, as we anticipated minimal motion artifacts. However, after examining the pre-processed data, we observed that the baseline shift in the raw data was replaced by a significant spike. We had hoped that spike correction combined with filtering would resolve this issue, but it did not.

Manual spike removal can be time-consuming and subjective, and the Satori software's interface is not ideal for this task, increasing the likelihood of compromising the signal quality.

CBSI, as proposed by (Cui et al., 2010), is based on the negative correlation between HbO and HbR concentrations. The method assumes that positively correlated features in the signal arise from motion and should be removed. However, (Balardin et al., 2017) demonstrated that only eyebrow movement significantly disrupts the expected negative correlation between oxy-Hb and deoxy-Hb.

TDDR is a motion correction technique based on robust regression, effectively removing baseline shift and spike artifacts. Moreover, (Fishburn et al., 2019) found that TDDR outperforms CBSI in activation detection, as CBSI relies on assumptions that do not always hold true concerning the relationships between HbO and HbR. However, TDDR has limitations: its effectiveness is significantly reduced when high-frequency components are present in the signal (Fishburn et al., 2019). Satori therefore filters out the higher frequencies before applying the TDDR.

Given the choice between manual motion correction, TDDR, or CBSI, we opted for TDDR. We also considered using spike correction to remove the high frequency spikes, but TDDR effectively corrects motion artifacts in frequencies up to 0.5 Hz (Fishburn et al., 2019), and our low-pass filter is set at 0.09 Hz, making TDDR suitable for our frequency range.

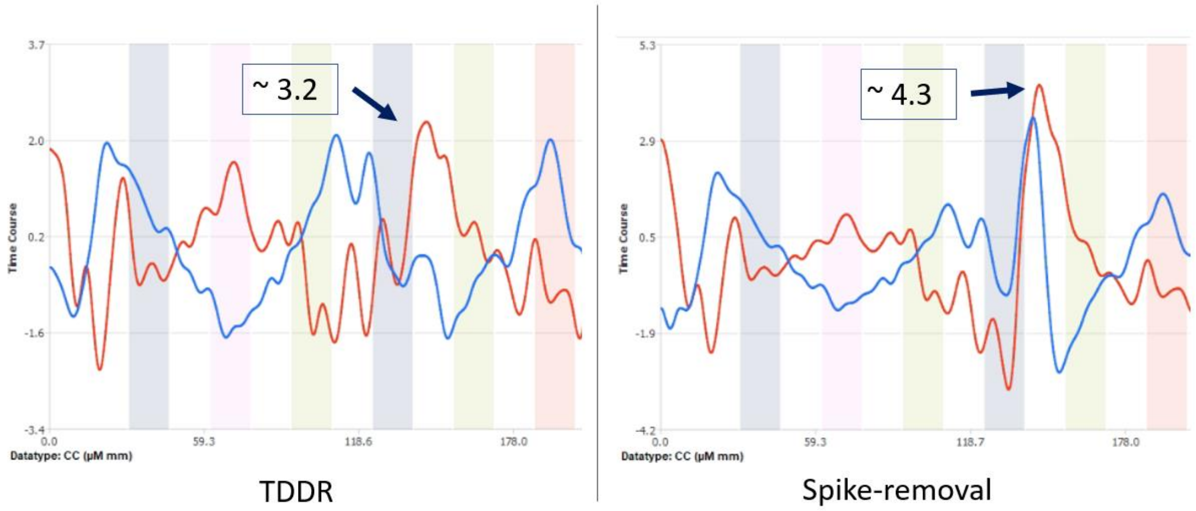


Figure 9-9 Comparative visualization of TDDR and spike-correction methods, illustrating the superior spike reduction achieved by TDDR. Data has been filtered and normalized to enhance the clarity of differences between the two techniques. Image made in Satori and edited in PowerPoint.

9.3.7 Physiological Noise removal

Physiological noise, such as heartbeat, respiration, and low-frequency content from blood pressure fluctuations, can contaminate neuroimaging data. These unwanted signals can obscure underlying neural activity, making data interpretation and accurate conclusions about brain function challenging (Klein & Kranczioch, 2019). Hence, removing physiological noise is crucial for improving neuroimaging studies' reliability and validity.

Satori provides an array of strategies to address the issue at hand, including Short-Channel Regression, Global Component Regression, and Temporal Filtering. In our study, we opted to only employ Temporal Filtering as the sole method for eliminating physiological noise. This approach effectively mitigates both low- and high-frequency noise by applying filters to the data, isolating the frequency range pertinent to our research, and subsequently eliminating undesired signals.

We refrained from utilizing Short-Channel Regression in our analysis, as our montage setup does not incorporate short-channels. This technique is advantageous when the montage includes short-channels, as it facilitates noise reduction by leveraging spatial information derived from these channels.

Moreover, we decided against implementing Global Component Regression. While this method can offer benefits under certain conditions, Satori advises its application in cases involving large montage setups with numerous channels, some of which may not be relevant to the study. Our montage configuration consists of a moderately-sized 16x16 arrangement, with all channels bearing significance to our research. Consequently, Global Component Regression was not considered the most appropriate choice for our analysis.

In selecting the appropriate cut-off frequencies for the filter, we followed the guidelines provided by (Pinti et al., 2019). The initial step involved determining the stimulation frequency to ensure it would not be filtered out. To accomplish this, we calculated the stimulation frequency by summing the duration of our stimulation block and the rest time. Given that the stimulation block's duration varied, we found the minimum and maximum time using the mean plus or minus the standard deviation, resulting in values of 15.73 seconds and 16.47 seconds, respectively. With a rest time of 15 seconds, we estimated the stimulus frequency range to be approximately [0.0317, 0.0325] Hz.

Subsequently, we pinpointed the specific frequencies that we aim to include and exclude. In cognitive task we want to look at changes in hemodynamic due to neurovascular coupling (Phillips et al., 2016), but there is physiological noise, such as cardiovascular oscillations that contaminate these changes. These include heartbeats (approximately 1 Hz), respiration (approximately 0.3 Hz), and Mayer waves (approximately 0.1 Hz), all which impact fNIRS data (Naseer & Hong, 2015; Pinti et al., 2019). Mayer waves represent spontaneous arterial blood pressure oscillations (Luke et al., 2021). Furthermore, three very low- cardiovascular - frequency oscillations, one at approximately 0.04 Hz linked to neurogenic activity in vessel walls and two others at approximately 0.01 and 0.007 Hz related to vascular endothelial function, should also be considered (Stefanovska, 2007).

Taking all these factors into account, we selected a frequency range of [0.01, 0.09] Hz, which aimed to encompass the highest passband while still eliminating the aforementioned frequency oscillations. We could not remove neurogenic frequency oscillation because it was too close to our stimulation frequency. Furthermore, we included the second harmonic of our stimulation frequency, which contains significant relevant information (Pinti et al., 2019). To achieve this, we employed a Gaussian smoothing low-pass filter with a cut-off frequency of 0.09 Hz and a Butterworth high-pass filter with a cut-off frequency of 0.01 Hz.

The Gaussian smoothing low-pass filter is employed to eliminate high-frequency noise while retaining lower-frequency content in the data. The Gaussian filter is preferred over Butterworth low-pass filters, because it maintains more frequencies in the data, which is crucial for subsequent general linear model (GLM) analysis (Brain-Inovation, 2023).

The Butterworth high-pass filter is used to remove low-frequency drifts in fNIRS data, which can vary considerably across subjects (Brain-Inovation, 2023). This filter allows for direct specification of the cut-off frequency in Hz and is implemented as a second-order filter in Satori. The Butterworth filter offers a smooth frequency response, ensuring that the filtered signal retains essential information relevant to the neural activity of interest (Brain-Inovation, 2023).

9.3.8 Normalization

In fNIRS data analysis, it is essential to normalize the data to enable meaningful comparisons across various channels and subjects. This is due to the significant differences in signal levels between channels, which arise from the physical and physiological properties of fNIRS measurements. Notably, the results of the general linear model (GLM), such as the significance of betas and contrasts, are not influenced by signal levels. However, normalizing data remains vital for comparing effect sizes across channels and conducting multi-subject analyses where signal levels can differ considerably among corresponding channels (Brain-Inovation, 2023).

For this study, we utilized the z-normalization technique. This method involves mean-centring the signal in a channel by subtracting the mean and relating it to the signal's standard deviation fluctuations. By representing signal fluctuations in units of standard deviation, the deviation of individual values or mean effects can be more easily understood and interpreted within standard statistical frameworks.

9.3.9 Finished pre-processed data

Here is a image showcasing the comparative visualization showcases the impact of applying Workflow 2 to preprocess the data. The figure demonstrates the removal of baseline drift through highpass filtering, illustrated by the black stippled line. As a result, the signal now fluctuates around the stippled line instead of being influenced by baseline drift. Additionally, the application of lowpass frequency filtering has significantly improved the clarity of the signal by removing rapid fluctuations. Also, the the baselinedrift and spikes has been removed with TDDR. Moreover, the z-normalization process has transformed the signal range from -190 to 190 to a narrower range of -2 to 2.5, enhancing comparability, and also it made the amplitude of HbR (blue) equal to HbO.

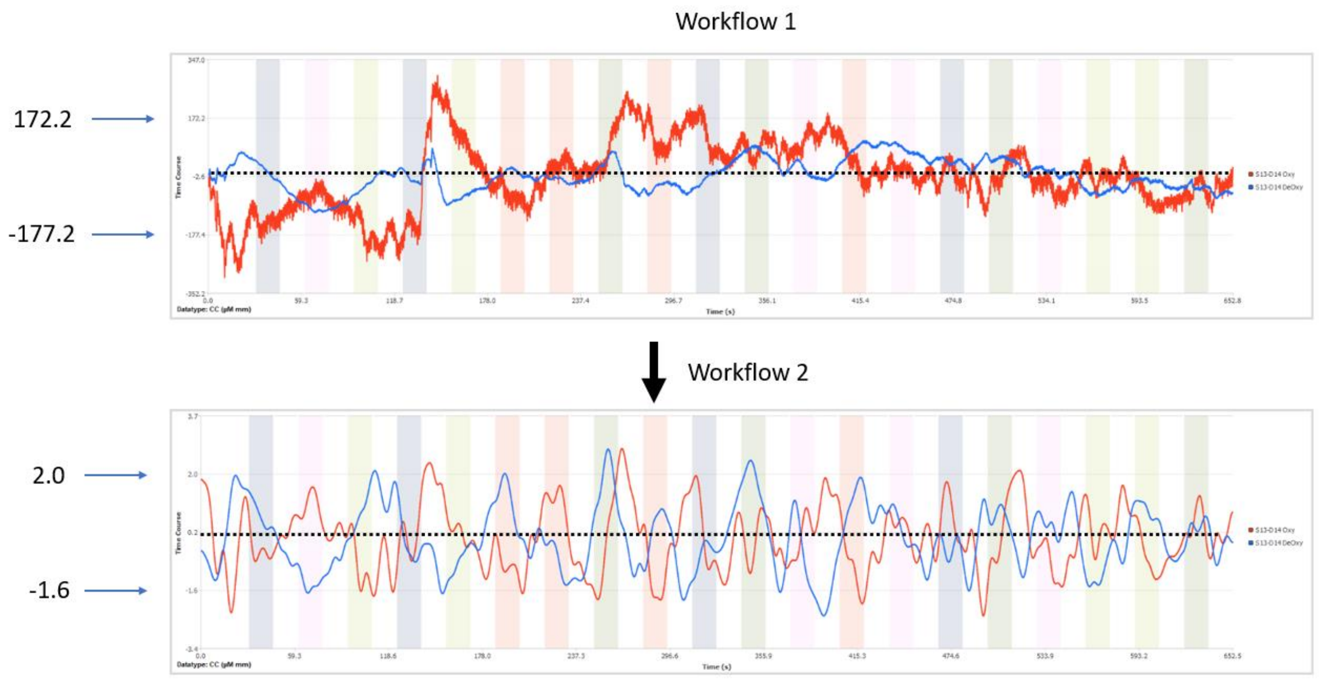


Figure 9-10 The comparative visualization showcases the impact of applying Workflow 2 to preprocess the data.

9.4 RESULTS FROM ANALYSIS

In our study, we analyzed the fNIRS data from 60 participants, each tested under 5 different conditions and with 42 fNIRS channels. Due to the large number of statistical tests performed (12,600 tests), we encountered the multiple comparisons problem, which could potentially increase the likelihood of false positives in our results.

To address the multiple comparisons problem in our data analysis, we applied the False Discovery Rate (FDR) correction method by (Benjamini & Hochberg, 1995). This method controls the proportion of false positives among the significant results, rather than the overall number of false positives, making it a suitable choice for fNIRS data analysis (Lührs et al., 2022).

By using the FDR correction, we aimed to identify truly significant channels while accounting for the multiple comparisons problem. The FDR method adapts to the amount of activity in the data and maintains a high sensitivity to detect true effects. As a result, we minimized the risk of false positives and ensured more accurate results in our fNIRS data analysis. With this approach, we were able to draw more reliable conclusions from our study.

We proceeded to create several contrast maps. A contrast map is a statistical map that highlights brain regions with significant differences or relationships between conditions. The contrast map is then generated by applying the contrast to the beta weights across all channels, resulting in a statistical value, t-value, for each location. In GLM contrasts, the '>' symbol compares the effects of two conditions. A positive t-value, colored red in 3D and 2D views, indicates the left side has a stronger effect, while a negative t-value, colored blue, means the right side has a stronger effect.

Our GLM analysis looked at 4 different contrast maps:

1. Happy/Neutral Congruent > Happy/Neutral Incongruent
   (Dot behind happy emotion vs. behind neutral emotion)

2. Fearful/Neutral Congruent > Fearful/Neutral Incongruent
   (Dot behind fearful emotion vs. behind neutral emotion)

3. Happy/Neutral Congruent + Fearful/Neutral Congruent >
   Happy/Neutral Incongruent + Fearful/Neutral Incongruent
   (Dot behind most negative emotion > Dot behind most positive emotion)

4. Happy/Neutral Incongruent + Fearful/Neutral Congruent >
   Happy/Neutral Congruent + Fearful/Neutral Incongruent
   (Dot behind emotional face > Dot behind neutral face)

We found a significant difference in contrast map number 2 and 4. Here are the result:

Contrast Map 2: This map identified a statistically significant difference in HbR concentration between two conditions: Fearful/Neutral Congruent trials > Fearful/Neutral Incongruent trials. The observed t-value of 2.544812 represents the magnitude of the difference between the conditions in terms of standard errors. The corresponding p-value of 0.01357, which is less than the commonly used threshold of 0.05, suggests that this difference is unlikely to have occurred by chance alone. In practical terms, this result implies that there is a higher HbR concentration in the right ventral medial PFC when the dot is located behind the fearful face compared to when it is behind the neutral face. See images bellow to see the contrast map in 3D and 2D view.

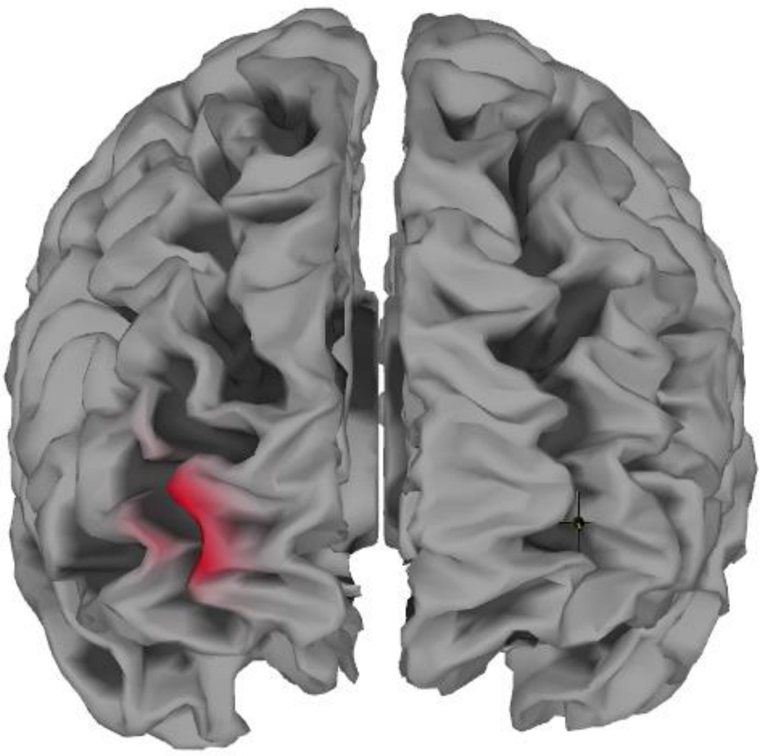


Figure 9-11 3D illustration of significant differences in contras map 2.

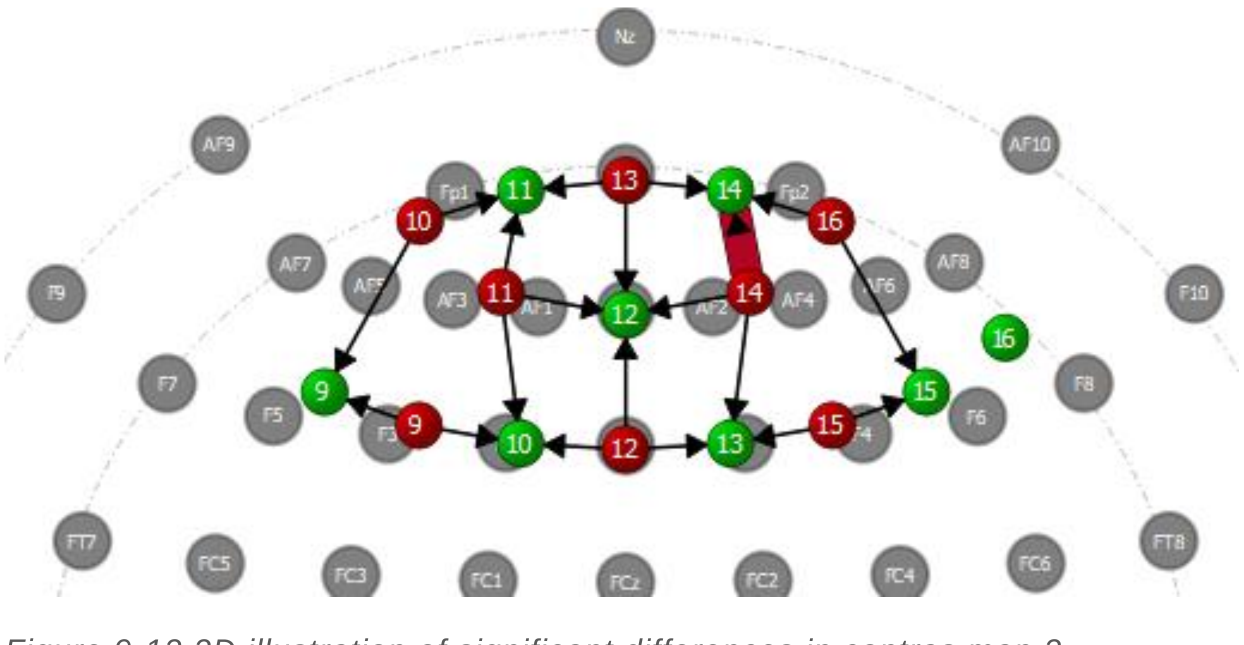


Figure 9-12 2D illustration of significant differences in contras map 2.

Contrast Map 4: This map revealed a statistically significant difference in HbR concentration at channel 15-15 between two conditions: Happy/Neutral Incongruent + Fearful/Neutral Congruent > Happy/Neutral Congruent + Happy/Neutral Incongruent. With a t-value of 2.961729 and a p-value of 0.004403 (which is below the commonly used threshold of 0.05), the observed difference is unlikely to be due to chance alone. In practical terms, this result indicates a higher HbR concentration in the right dorsal PFC when the dot is positioned behind a face expressing emotion compared to when it is behind a neutral face. See images bellow to see the contrast map in 3D and 2D view.

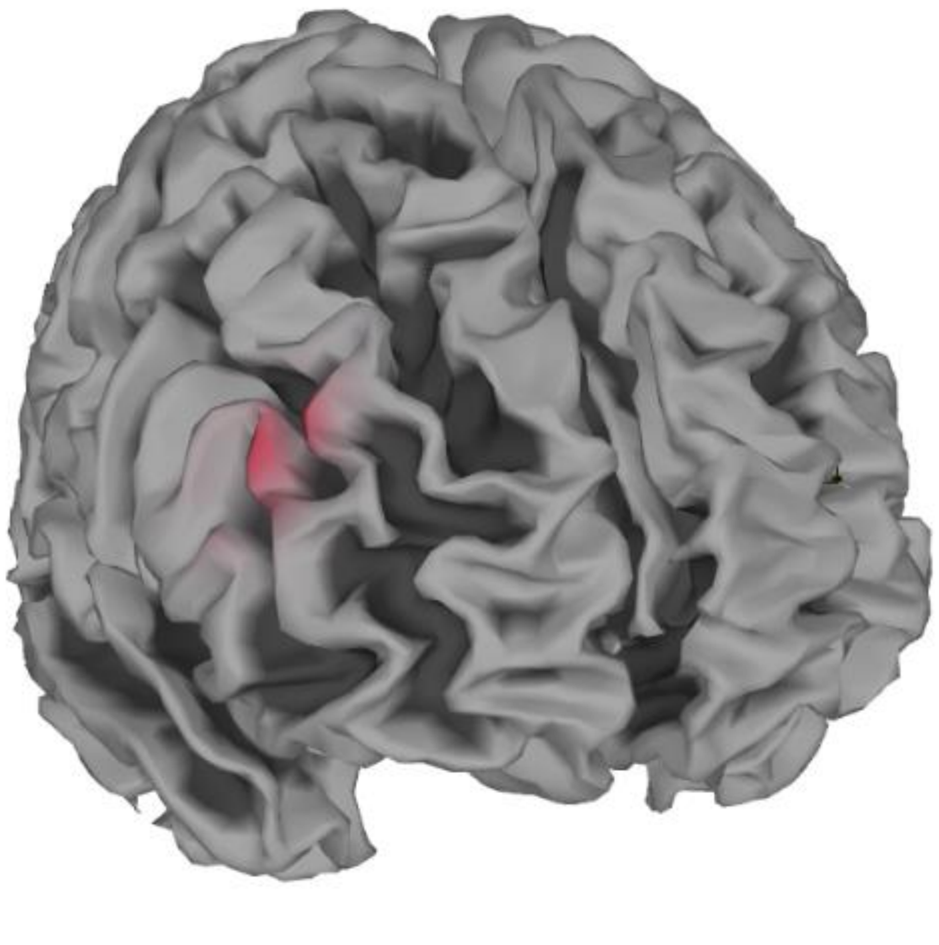


Figure 9-13 3D illustration of significant differences in contras map 4.

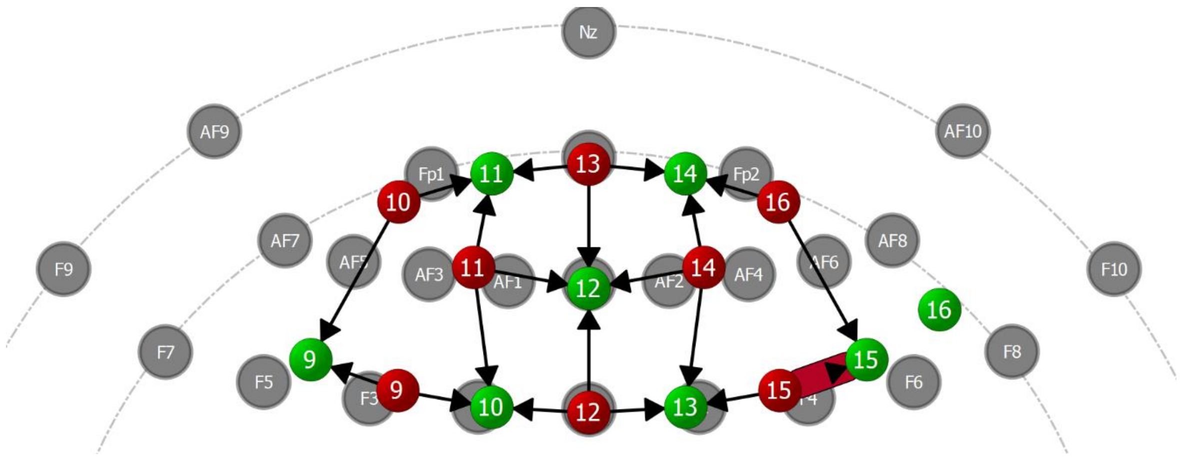


Figure 9-14 2D illustration of significant differences in contras map 4.

10 CONCLUSION

This master's thesis has successfully achieved its objectives by establishing a state-of-the-art test laboratory, setting up a hybrid fNIRS/ET and hybrid EEG/ET system, designing an effective dot-probe experiment paradigm, and creating a comprehensive experiment procedure pipeline. The research was further strengthened by the large-scale experiment conducted using the combined approach of the two hybrid systems with the dot-probe task. Additionally, the study has developed a straightforward pipeline for fNIRS data pre-processing.

The results of the multi-subject GLM analysis revealed two significant findings. First, there is a higher HbR concentration in the left orbitofrontal/ventral medial prefrontal cortex when the dot is located behind the fearful face compared to when it is behind the neutral face. Second, there is a higher HbR concentration in the left dorsal/ventral medial prefrontal cortex when the dot is positioned behind a face expressing emotion compared to when it is behind a neutral face.

While the results are interesting, there is still a lot more to learn in this area. However, the progress made in this thesis shows that using fNIRS to study AB can help us get a better understanding of the underlying neural processes.

11 FUTURE WORK

While the current study has provided some valuable insights into the neural mechanisms of AB by employing fNIRS systems, its main focus was to lay ground for future research. There are several promising avenues for future research that could further enhance our understanding of AB.

1. Correlation analysis between hybrid fNIRS, ET data, and reaction time data: As a next step, it would be beneficial to perform an analysis that investigates the correlation between the hybrid fNIRS and ET data, and reaction time data on each participant. This could shed light on the relationship between neural activation, gaze patterns, and response times, and potentially reveal specific patterns of neural and behavioural activity that could serve as markers for AB.

2. Correlation analysis between hybrid EEG, ET data, and reaction time data: Similarly, an analysis could be conducted that examines the correlation between the hybrid EEG and ET data, and reaction time data on each participant. This would allow for a more comprehensive understanding of the electrocortical dynamics underlying AB and the interplay between neural activity, gaze behaviour, and response times.

3. Cross-modal correlation analysis of EEG and fNIRS signals using ET data as common ground: To further investigate the neural mechanisms of AB, it would be valuable to examine the correlation between the EEG and fNIRS signals, using the ET data as a common ground between them. This analysis could enable the identification of potential associations between the two modalities and allow for a more in-depth understanding of the underlying neural processes that contribute to AB.

12 References

Althobaiti, M., & Al-Naib, I. (2020). Recent Developments in Instrumentation of Functional Near-Infrared Spectroscopy Systems. *Applied Sciences*, *10*(18), 6522. https://doi.org/10.3390/app10186522

Andrzejewski, J. A., & Carlson, J. M. (2020). Electrocortical responses associated with attention bias to fearful facial expressions and auditory distress signals. *International Journal of Psychophysiology*, *151*, 94-102.

Armstrong, T., & Olatunji, B. O. (2012). Eye tracking of attention in the affective disorders: a meta-analytic review and synthesis. *Clin Psychol Rev*, *32*(8), 704-723. https://doi.org/10.1016/j.cpr.2012.09.004

Armstrong, T., & Olatunji, B. O. (2012). Eye tracking of attention in the affective disorders: A meta-analytic review and synthesis. *Clinical Psychology Review*, *32*(8), 704-723. https://doi.org/10.1016/j.cpr.2012.09.004

Balardin, J. B., Morais, G. A. Z., Furucho, R. A., Trambaiolli, L. R., & Sato, J. R. (2017). Impact of communicative head movements on the quality of functional near-infrared spectroscopy signals: negligible effects for affirmative and negative gestures and consistent artifacts related to raising eyebrows. *Journal of biomedical optics*, *22*(4), 046010. https://doi.org/10.1117/1.jbo.22.4.046010

Barry, T. J., Vervliet, B., & Hermans, D. (2015). An integrative review of attention biases and their contribution to treatment for anxiety disorders [Review]. *Frontiers in Psychology*, *6*. https://doi.org/10.3389/fpsyg.2015.00968

Benjamini, Y., & Hochberg, Y. (1995). Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological)*, *57*(1), 289-300.

Black, K. (2022). What is the difference between EEG and an event related potential (ERP)? In. Brainstuff.org: Brainstuff.

Blasi, A., Lloyd-Fox, S., H. Johnosn, M., & Elwell, C. (2014). Test–retest reliability of functionalnear infrared spectroscopy in infants. https://doi.org/10.1117/1.nph.1.2.025005.full

Bradley, B. P., Mogg, K., & Millar, N. H. (2000). Covert and overt orienting of attention to emotional faces in anxiety. *Cognition & Emotion*, *14*(6), 789-808.

Brain-Inovation. (2023). Satori User Manual. In (1.8.2 ed.).

Brigadoi, S., Ceccherini, L., Cutini, S., Scarpa, F., Scatturin, P., Selb, J., Gagnon, L., Boas, D. A., & Cooper, R. J. (2014). Motion artifacts in functional near-infrared spectroscopy: A comparison of motion correction techniques applied to real cognitive data. *Neuroimage*, *85*, 181-191. https://doi.org/10.1016/j.neuroimage.2013.04.082

Britton, J. C., Bar-Haim, Y., Clementi, M. A., Sankin, L. S., Chen, G., Shechner, T., Norcross, M. A., Spiro, C. N., Lindstrom, K. M., & Pine, D. S. (2013). Training-associated changes and stability of attention bias in youth: Implications for Attention Bias Modification Treatment for pediatric anxiety. *Developmental Cognitive Neuroscience*, *4*, 52-64.

Brown, H., Eley, T., Broeren, S., Macleod, C., Rinck, M., Hadwin, J., & Lester, K. (2014). Psychometric properties of reaction time based experimental paradigms measuring anxiety-related information-processing biases in children. *Journal of anxiety disorders*, *28*(1), 97-107.

Carlson, J. M. (2021). A systematic review of event-related potentials as outcome measures of attention bias modification. *Psychophysiology*, *58*(6). https://doi.org/10.1111/psyp.13801

Carlson, J. M., Cha, J., Harmon-Jones, E., Mujica-Parodi, L. R., & Hajcak, G. (2014). Influence of the BDNF genotype on amygdalo-prefrontal white matter microstructure is linked to nonconscious attention bias to threat. *Cerebral Cortex*, *24*(9), 2249-2257.

Carlson, J. M., Cha, J., & Mujica-Parodi, L. R. (2013). Functional and structural amygdala–anterior cingulate connectivity correlates with attentional bias to masked fearful faces. *Cortex*, *49*(9), 2595-2600.

Carlson, J. M., & Fang, L. (2020). The stability and reliability of attentional bias measures in the dot-probe task: Evidence from both traditional mean bias scores and trial-level bias scores. *Motivation and Emotion*, *44*(5), 657-669. https://doi.org/10.1007/s11031-020-09834-6

Carlson, J. M., Reinke, K. S., LaMontagne, P. J., & Habib, R. (2011). Backward masked fearful faces enhance contralateral occipital cortical activity for visual targets within the spotlight of attention. *Social Cognitive and Affective Neuroscience*, *6*(5), 639-645.

Carter, B. T., & Luke, S. G. (2020). Best practices in eye tracking research. *International Journal of Psychophysiology*, *155*, 49-62.

Chew, P. (2015). Attentional bias: a methodological review. *Education Sciences and Psychology*, *5*, 14-29.

Cisler, J. M., & Koster, E. H. W. (2010). Mechanisms of attentional biases towards threat in anxiety disorders: An integrative review. *Clinical Psychology Review*, *30*(2), 203-216. https://doi.org/10.1016/j.cpr.2009.11.003

Craske, M. G. (2012). Transdiagnostic treatment for anxiety and depression. *Depression and Anxiety*.

Cui, X., Bray, S., & Reiss, A. L. (2010). Functional near infrared spectroscopy (NIRS) signal improvement based on negative correlation between oxygenated and deoxygenated hemoglobin dynamics. *Neuroimage*, *49*(4), 3039-3046. https://doi.org/10.1016/j.neuroimage.2009.11.050

De Ruiter, C., & Brosschot, J. F. (1994). The emotional Stroop interference effect in anxiety: attentional bias or cognitive avoidance? *Behaviour research and therapy*, *32*(3), 315-319.

Delpy, D. T., Cope, M., van der Zee, P., Arridge, S., Wray, S., & Wyatt, J. (1988). Estimation of optical pathlength through tissue from direct time of flight measurement. *Physics in Medicine & Biology*, *33*(12), 1433.

Disner, S. G., Beevers, C. G., Haigh, E. A., & Beck, A. T. (2011). Neural mechanisms of the cognitive model of depression. *Nature Reviews Neuroscience*, *12*(8), 467-477.

Dobson, K. S., & Dozois, D. J. (2004). Attentional biases in eating disorders: A meta-analytic review of Stroop performance. *Clinical Psychology Review*, *23*(8), 1001-1022.

Duff, K., Beglinger, L. J., Schultz, S. K., Moser, D. J., McCaffrey, R. J., Haase, R. F., Westervelt, H. J., Langbehn, D. R., Paulsen, J. S., & Group, H. s. S. (2007). Practice effects in the prediction of long-term cognitive outcome in three patient samples: A novel prognostic index. *Archives of Clinical Neuropsychology*, *22*(1), 15-24.

Duque, A., & Vázquez, C. (2015). Double attention bias for positive and negative emotional faces in clinical depression: Evidence from an eye-tracking study. *Journal of behavior therapy and experimental psychiatry*, *46*, 107-114. https://doi.org/https://doi.org/10.1016/j.jbtep.2014.09.005

Ehlis, A.-C., Schneider, S., Dresler, T., & Fallgatter, A. J. (2014). Application of functional near-infrared spectroscopy in psychiatry. *Neuroimage*, *85*, 478-488.

Eysenck, M. W., Derakshan, N., Santos, R., & Calvo, M. G. (2007). Anxiety and cognitive performance: attentional control theory. *Emotion*, *7*(2), 336.

Fishburn, F. A., Ludlum, R. S., Vaidya, C. J., & Medvedev, A. V. (2019). Temporal Derivative Distribution Repair (TDDR): A motion correction method for fNIRS. *Neuroimage*, *184*, 171-179. https://doi.org/10.1016/j.neuroimage.2018.09.025

Fox, E., Russo, R., Bowles, R., & Dutton, K. (2001). Do threatening stimuli draw or hold visual attention in subclinical anxiety? *J Exp Psychol Gen*, *130*(4), 681-700.

Friston, K. J., Holmes, A. P., Worsley, K. J., Poline, J. P., Frith, C. D., & Frackowiak, R. S. (1994). Statistical parametric maps in functional imaging: a general linear approach. *Human Brain Mapping*, *2*(4), 189-210.

Garland, E. L., & Howard, M. O. (2014). A transdiagnostic perspective on cognitive, affective, and neurobiological processes underlying human suffering. *Research on Social Work Practice*, *24*(1), 142-151.

Gotlib, I. H., & McCann, C. D. (1984). Construct accessibility and depression: an examination of cognitive and affective factors. *Journal of personality and social psychology*, *47*(2), 427.

Gupta, R. S., Kujawa, A., & Vago, D. R. (2019). The neural chronometry of threat-related attentional bias: Event-related potential (ERP) evidence for early and late stages of selective attentional processing. *Int J Psychophysiol*, *146*, 20-42. https://doi.org/10.1016/j.ijpsycho.2019.08.006

Hilland, E., Landrø, N. I., Harmer, C. J., Browning, M., Maglanoc, L. A., & Jonassen, R. (2020). Attentional bias modification is associated with fMRI response toward negative stimuli in individuals with residual depression: a randomized controlled trial. *Journal of Psychiatry and Neuroscience*, *45*(1), 23-33. https://doi.org/10.1503/jpn.180118

Jahani, S., Setarehdan, S. K., Boas, D. A., & Yücel, M. A. (2018). Motion artifact detection and correction in functional near-infrared spectroscopy: a new hybrid method based on spline interpolation method and Savitzky–Golay filtering. *Neurophotonics*, *5*(1), 015003-015003.

Jiang, M. Y. W., & Vartanian, L. R. (2018). A review of existing measures of attentional biases in body image and eating disorders research. *Australian Journal of Psychology*, *70*(1), 3-17. https://doi.org/10.1111/ajpy.12161

Kappenman, E. S., Farrens, J. L., Luck, S. J., & Proudfit, G. H. (2014). Behavioral and ERP measures of attentional bias to threat in the dot-probe task: Poor reliability and lack of correlation with anxiety. *Frontiers in Psychology*, *5*, 1368.

Kappenman, E. S., MacNamara, A., & Proudfit, G. H. (2015). Electrocortical evidence for rapid allocation of attention to threat in the dot-probe task. *Social Cognitive and Affective Neuroscience*, *10*(4), 577-583.

Khan, H., Naseer, N., Yazidi, A., Eide, P. K., Hassan, H. W., & Mirtaheri, P. (2021). Analysis of Human Gait Using Hybrid EEG-fNIRS-Based BCI System: A Review [Review]. *Frontiers in Human Neuroscience*, *14*. https://doi.org/10.3389/fnhum.2020.613254

Khan, H., Noori, F. M., Yazidi, A., Uddin, M. Z., Khan, M. N. A., & Mirtaheri, P. (2021). Classification of Individual Finger Movements from Right Hand Using fNIRS Signals. *Sensors*, *21*(23), 7943. https://doi.org/10.3390/s21237943

Klein, F., & Kranczioch, C. (2019). Signal processing in fNIRS: a case for the removal of systemic activity for single trial data. *Frontiers in Human Neuroscience*, *13*, 331.

Krishnamoorthy-Natarajan, G., & Koide, M. (2016). BK channels in the vascular system. *International Review of Neurobiology*, *128*, 401-438.

Liu, H., Ivanov, K., Wang, Y., & Wang, L. (2015). A novel method based on two cameras for accurate estimation of arterial oxygen saturation. *BioMedical Engineering OnLine*, *14*, 1-17.

Liu, Z., Shore, J., Wang, M., Yuan, F., Buss, A., & Zhao, X. (2021). A systematic review on hybrid EEG/fNIRS in brain-computer interface. *Biomedical Signal Processing and Control*, *68*, 102595.

Lloyd-Fox, S., Blasi, A., & Elwell, C. (2010). Illuminating the developing brain: the past, present and future of functional near infrared spectroscopy. *Neuroscience & Biobehavioral Reviews*, *34*(3), 269-284.

Luck, S. J. (2014). *An introduction to the event-related potential technique*. MIT press.

Lührs, M., Manferlotti, E., & Heinecke, A. (2022). Satori Multi-Subject GLM Guide.

Luke, R., Shader, M. J., & McAlpine, D. (2021). Characterization of Mayer-wave oscillations in functional near-infrared spectroscopy using a physiologically informed model of the neural power spectra. *Neurophotonics*, *8*(4), 041001-041001.

Mansell, W., Harvey, A., Watkins, E. R., & Shafran, R. (2008). Cognitive behavioral processes across psychological disorders: A review of the utility and validity of the transdiagnostic approach. *International Journal of Cognitive Therapy*, *1*(3), 181-191.

Mattia, J. I., Heimberg, R. G., & Hope, D. A. (1993). The revised Stroop color-naming task in social phobics. *Behaviour research and therapy*, *31*(3), 305-313.

Meryem A. Yücel, A. v. L., Felix Scholkmann, Judit Gervain, Ippeita Dan, Hasan Ayaz, David Boas, Robert J. Cooper, Joseph Culver, Clare E. Elwell, Adam Eggebrecht, Maria A. Franceschini, Christophe Grova, Fumitaka Homae, Frédéric Lesage, Hellmuth Obrig, Ilias Tachtsidis, Sungho Tak, Yunjie Tong, Alessandro Torricelli, Heidrun Wabnitz, Martin Wolf. (2021). Best practices for fNIRS publications. https://doi.org/10.1117/1.nph.8.1.012101.short

Mogg, K., Garner, M., & Bradley, B. P. (2007). Anxiety and orienting of gaze to angry and fearful faces. *Biological psychology*, *76*(3), 163-169.

Monk, C. S., Nelson, E. E., McClure, E. B., Mogg, K., Bradley, B. P., Leibenluft, E., Blair, R. J. R., Chen, G., Charney, D. S., Ernst, M., & Pine, D. S. (2006). Ventrolateral Prefrontal Cortex Activation and

Attentional Bias in Response to Angry Faces in Adolescents With Generalized Anxiety Disorder. *American Journal of Psychiatry*, *163*(6), 1091-1097. https://doi.org/10.1176/ajp.2006.163.6.1091

Morris, J. S., Frith, C. D., Perrett, D. I., Rowland, D., Young, A. W., Calder, A. J., & Dolan, R. J. (1996). A differential neural response in the human amygdala to fearful and happy facial expressions. *Nature*, *383*(6603), 812-815. https://doi.org/10.1038/383812a0

Naim, R., Abend, R., Wald, I., Eldar, S., Levi, O., Fruchter, E., Ginat, K., Halpern, P., Sipos, M. L., Adler, A. B., Bliese, P. D., Quartana, P. J., Pine, D. S., & Bar-Haim, Y. (2015). Threat-Related Attention Bias Variability and Posttraumatic Stress. *American Journal of Psychiatry*, *172*(12), 1242-1250. https://doi.org/10.1176/appi.ajp.2015.14121579

Naseer, N., & Hong, K.-S. (2015). fNIRS-based brain-computer interfaces: a review. *Frontiers in Human Neuroscience*, *9*. https://doi.org/10.3389/fnhum.2015.00003

NIRx. (2017). Troubleshooting Signal Quality Getting Started Guide.

NIRx. (2018). NIRStar™ 15.2 User Manual. 14-142. www.nirx.net

NIRx. (2019). NIRSCap User Guide.

Nogueira, M. G., Silvestrin, M., Barreto, C. S. F., Sato, J. R., Mesquita, R. C., Biazoli, C., & Baptista, A. F. (2022). Differences in brain activity between fast and slow responses on psychomotor vigilance task: an fNIRS study. *Brain Imaging and Behavior*, *16*(4), 1563-1574. https://doi.org/10.1007/s11682-021-00611-8

Okazaki, Y., Abrahamyan, A., Stevens, C. J., & Ioannides, A. A. (2010). Wired for Her Face? Male Attentional Bias for Female Faces. *Brain Topography*, *23*(1), 14-26. https://doi.org/10.1007/s10548-009-0112-7

Peckham, A. D., McHugh, R. K., & Otto, M. W. (2010). A meta-analysis of the magnitude of biased attention in depression. *Depress Anxiety*, *27*(12), 1135-1142. https://doi.org/10.1002/da.20755

Pessoa, L., Kastner, S., & Ungerleider, L. G. (2002). Attentional control of the processing of neutral and emotional stimuli. *Cognitive Brain Research*, *15*(1), 31-45.

Pfabigan, D. M., Lamplmayr-Kragl, E., Pintzinger, N. M., Sailer, U., & Tran, U. S. (2014). Sex differences in event-related potentials and attentional biases to emotional facial stimuli. *Frontiers in Psychology*, *5*, 1477.

Phillips, A. A., Chan, F. H., Zheng, M. M. Z., Krassioukov, A. V., & Ainslie, P. N. (2016). Neurovascular coupling in humans: Physiology, methodological advances and clinical implications. *Journal of Cerebral Blood Flow & Metabolism*, *36*(4), 647-664. https://doi.org/10.1177/0271678x15617954

Pinti, P., Scholkmann, F., Hamilton, A., Burgess, P., & Tachtsidis, I. (2019). Current Status and Issues Regarding Pre-processing of fNIRS Neuroimaging Data: An Investigation of Diverse Signal Filtering Methods Within a General Linear Model Framework. *Frontiers in Human Neuroscience*, *12*. https://doi.org/10.3389/fnhum.2018.00505

Piper, S. K., Krueger, A., Koch, S. P., Mehnert, J., Habermehl, C., Steinbrink, J., Obrig, H., & Schmitz, C. H. (2014). A wearable multi-channel fNIRS system for brain imaging in freely moving subjects. *Neuroimage*, *85*, 64-71. https://doi.org/10.1016/j.neuroimage.2013.06.062

Pourtois, G., Schwartz, S., Seghier, M. L., Lazeyras, F., & Vuilleumier, P. (2006). Neural systems for orienting attention to the location of threat signals: an event-related fMRI study. *Neuroimage*, *31*(2), 920-933.

Price, R. B., Kuckertz, J. M., Siegle, G. J., Ladouceur, C. D., Silk, J. S., Ryan, N. D., Dahl, R. E., & Amir, N. (2015). Empirical recommendations for improving the stability of the dot-probe task in clinical research. *Psychological Assessment*, *27*(2), 365-376. https://doi.org/10.1037/pas0000036

Price, R. B., Siegle, G. J., Silk, J. S., Ladouceur, C. D., McFarland, A., Dahl, R. E., & Ryan, N. D. (2014). LOOKING UNDER THE HOOD OF THE DOT-PROBE TASK: AN fMRI STUDY IN ANXIOUS YOUTH. *Depression and Anxiety*, *31*(3), 178-187. https://doi.org/10.1002/da.22255

Quaresima, V., & Ferrari, M. (2019). Functional Near-Infrared Spectroscopy (fNIRS) for Assessing Cerebral Cortex Function During Human Behavior in Natural/Social Situations: A Concise Review. *Organizational Research Methods*, *22*(1), 46-68. https://doi.org/10.1177/1094428116658959

Reddy, P., Izzetoglu, M., Shewokis, P. A., Sangobowale, M., Diaz-Arrastia, R., & Izzetoglu, K. (2021). Evaluation of fNIRS signal components elicited by cognitive and hypercapnic stimuli. *Scientific Reports*, *11*(1). https://doi.org/10.1038/s41598-021-02076-7

Reutter, M., Hewig, J., Wieser, M. J., & Osinsky, R. (2017). The N2pc component reliably captures attentional bias in social anxiety. *Psychophysiology*, *54*(4), 519-527. https://doi.org/10.1111/psyp.12809

Rogers, D., Murphy, E., Winders, S. J., & Greene, C. (2020). Attentional Bias Components in Anxiety and Depression: A Systematic Review.

Sass, S. M., Heller, W., Fisher, J. E., Silton, R. L., Stewart, J. L., Crocker, L. D., Edgar, J. C., Mimnaugh, K. J., & Miller, G. A. (2014). Electrophysiological evidence of the time course of attentional bias in non-patients reporting symptoms of depression with and without co-occurring anxiety. *Frontiers in Psychology*, *5*, 301.

Schmukle, S. C. (2005). Unreliability of the dot probe task. *European Journal of Personality*, *19*(7), 595-605.

Scholkmann, F., & Wolf, M. (2012). Measuring brain activity using functional near infrared spectroscopy: a short review. *Spectroscopy Europe*.

Scholkmann, F. W., Martin. (2012). Measuring brain activity using functional near infrared spectroscopy: a short review. *Spectroscopy Europe*.

Seidel, O., Carius, D., Roediger, J., Rumpf, S., & Ragert, P. (2019). Changes in neurovascular coupling during cycling exercise measured by multi-distance fNIRS: a comparison between endurance athletes and physically active controls. *Experimental Brain Research*, *237*(11), 2957-2972. https://doi.org/10.1007/s00221-019-05646-4

SR-research. (2017). EyeLinkPortable Duo User Manual. https://www.manualslib.com/manual/2114770/Sr-Research-Eyelink-Portable-Duo.html#manual

SR-Research. (2023). *Trackable Range Calculator*. https://www.sr-research.com/trackable-range-calculator/

Staugaard, S. R. (2009). Reliability of two versions of the dot-probe task using photographic faces. *Psychology Science Quarterly*, *51*(3), 339-350.

Stefanovska, A. (2007). Coupled Oscillatros: Complex But Not Complicated Cardiovascular and Brain Interactions. *IEEE Engineering in Medicine and Biology Magazine*, *26*(6), 25-29. https://doi.org/10.1109/emb.2007.907088

Stevens, S., Rist, F., & Gerlach, A. L. (2011). Eye movement assessment in individuals with social phobia: Differential usefulness for varying presentation times? *Journal of behavior therapy and experimental psychiatry*, *42*(2), 219-224.

Suresh, K. (2011). An overview of randomization techniques: An unbiased assessment of outcome in clinical research. *J Hum Reprod Sci*, *4*(1), 8-11. https://doi.org/10.4103/0974-1208.82352

Torrence, R. D. (2015). PREFRONTAL CORTEX ACTIVITY DURING ATTENTIONAL BIAS CONDITIONING WITH FEARFUL FACES: A NEAR-INFRARED SPECTROSCOPY ANALYSIS.

Torrence, R. D., & Troup, L. J. (2018). Event-related potentials of attentional bias toward faces in the dot-probe task: A systematic review. *Psychophysiology*, *55*(6), e13051. https://doi.org/10.1111/psyp.13051

Torricelli, A., Contini, D., Pifferi, A., Caffini, M., Re, R., Zucchelli, L., & Spinelli, L. (2014). Time domain functional NIRS imaging for human brain mapping. *Neuroimage*, *85*, 28-50. https://doi.org/10.1016/j.neuroimage.2013.05.106

Usakli, A. B. (2010). Improvement of EEG Signal Acquisition: An Electrical Aspect for State of the Art of Front End. *Computational Intelligence and Neuroscience*, *2010*, 1-7. https://doi.org/10.1155/2010/630649

Waechter, S., Nelson, A. L., Wright, C., Hyatt, A., & Oakman, J. (2014). Measuring Attentional Bias to Threat: Reliability of Dot Probe and Eye Movement Indices. *Cognitive Therapy and Research*, *38*(3), 313-333. https://doi.org/10.1007/s10608-013-9588-2

Weierich, M. R., Treat, T. A., & Hollingworth, A. (2008). Theories and measurement of visual attentional processing in anxiety. *Cognition and emotion*, *22*(6), 985-1018.

WHO, V., Sergey. (2023). *Mental Health*. World Health Organisation. https://www.who.int/health-topics/mental-health#tab=tab_2

Williams, J. M. G., Mathews, A., & MacLeod, C. (1996). The emotional Stroop task and psychopathology. *Psychological bulletin*, *120*(1), 3.

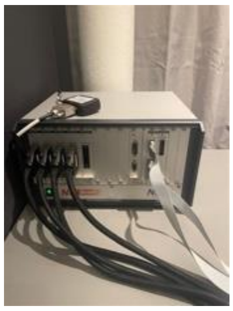
Yuan, Y., Li, G., Ren, H., & Chen, W. (2021). Effect of Light on Cognitive Function During a Stroop Task Using Functional Near-Infrared Spectroscopy. *Phenomics*, *1*(2), 54-61. https://doi.org/10.1007/s43657-021-00010-5

Zhang, Y. F., Lasfargue, A., & Berry, I. (2021). *Auditory cortex activation is modulated nonlinearly by stimulation duration: A functional near-infrared spectroscopy (fNIRS) study*. Cold Spring Harbor Laboratory. https://dx.doi.org/10.1101/2021.08.02.454752

13 APPENDIX

A.1 STEP-BY-STEP GUIDE FOR EXPERIMENT PROCEDURE

BEFORE PARTICIPANT ARRIVE



1) Turn on equipment
   - ☐ "Cyber power" (black box under desk)
   - ☐ NIRScout [se photo] 10-15min before experiment start (ON-button is the green button on lower left corner)
2) Prepare participant folder on experimenter PC (to the left), password: infrared
   - ☐ Go to desktop → Data → "participant ID"
   - ☐ Fill out experiment log
     - ☐ Check "Oversikt deltakere" on teams for age and participantID
3) Make sure you have this equipment ready (hint to code: Bond)

EEG EQUIPMENT

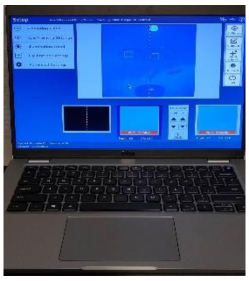
- ☐ EEG-CAP, AMPLIFYER & BASESTATION
- ☐ BOX FOR CLEANING EEG
- ☐ SHAMPOO,CONDITIONER & HAIRDRYER
- ☐ TOOTHBRUSH
- ☐ GEL AND SYRINGES
- ☐ MARKER (RED LIPLINER)

FNIRS EQUIPMENT

- ☐ FNIRS CAPS (54, 56 AND 58CM), NOTE WHICH CAP SIZE IS MOUNTED. WE MIGHT NEED TO CHANGE MONTAGE DURING EXPERIMENT
- ☐ SHOWERCAP
- ☐ HEADLIGHT

4) Set up fNIRS software
   - ☐ On experimenter PC (to the left)
   - ☐ Start program called NIRx NIRStar 15.2
   - ☐ Make sure the program is connected to the NIRScout. It should say online in the upper right corner. (If it says offline, try restart the program)
   - ☐ Double check if the right montage is selected.
     Configure Hardware --> Predefined Montages --> Choose montage: OccPrefr_16x16

- ☐ Choose storage directory
  File Options --> choose desktop\Data\ Participant ID .
  Name the file “[participant ID]_fnirs” --> click save
- ☐ Change file prefix
  Name the file prefix “[ParticipantID]_NIRS”
- ☐ Make sure exports data to Homer2 format is checked
  Click “ok”
- ☐ Write the participants’ age
  Configure hardware --> [Hb] Parametes --> Subject age. Click “ok



5) Set up eye-tracking
- ☐ Turn on the host-pc (laptop on the right)
  - ☐ After 30-60s the display should look like this [see photo]. That means the eye-tracker is ready for use.

- ☐ Take light measurement
  - ☐ Close curtains and close the door
  - ☐ Put the light measure on the headstand and face the sensor towards the screen
  - ☐ Make sure you are not covering the light with your body
  - ☐ Measurements should be between 70-140 (usually its 130±5)
  - ☐ If light measure over threshold, note it in experiment log

6) Prepare syringes with gel

7) Have wireless keyboard turned on and placed on the participant table

8) Check that fNIRS caps looks fine (all 3 of them):

- ☐ If the plastic clips are broken (the plastic holding the probes together):
  - ☐ Replace it.

  Note: Spare parts are in bag [see photo]

WHEN PARTICIPANT ARRIVE

9) One experimenter meets participant and follow them to the lab
10) Let them leave their stuff in the closets outside of the room
    - ☐ Turn off phone / watch
    - ☐ Take off earrings (if they are big)
    - ☐ Lock closet
    - ☐ Make sure they do not have makeup, if they do give them makeup remover.
11) Read these instructions:

Welcome to the experiment and thank you for participating. We will be in this room where we will conduct three different measurements, eyetracking, fNIRS and EEG. You will be given the opportunity to use the restroom between the tasks. If you need to use the restroom now, please let me know

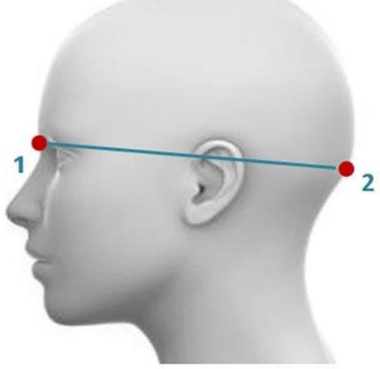
Before we begin, I need to make sure you have signed the consent form and filled out questionnaires in Nettskjema.

- o If not, don't worry, you can do it now. The experiment will be a bit delayed because of this (it takes 10-15 minutes to fill out the forms). You should have received a link to the "Samtykkeskjema" via mail.

Any questions?

let us begin with the first measurement: eye-tracking and fNIRS

PREPARATIONS FOR FNIRS



12) Choose cap size
- ☐ Ask participant to sit down on the chair
- ☐ Measure around the head including inion and middle between eyebrows [se photo]
- ☐ Choose cap size 54, 56 OR 58cm. *If between sizes: Always round to closest size, preferably down!*

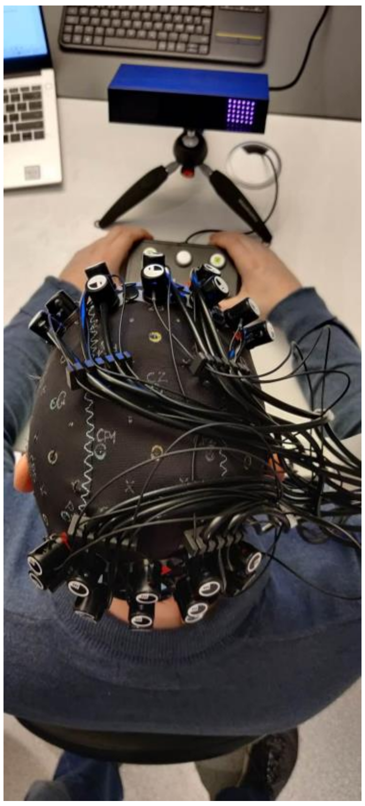
13) Write down **head circumference** size in the experimental log

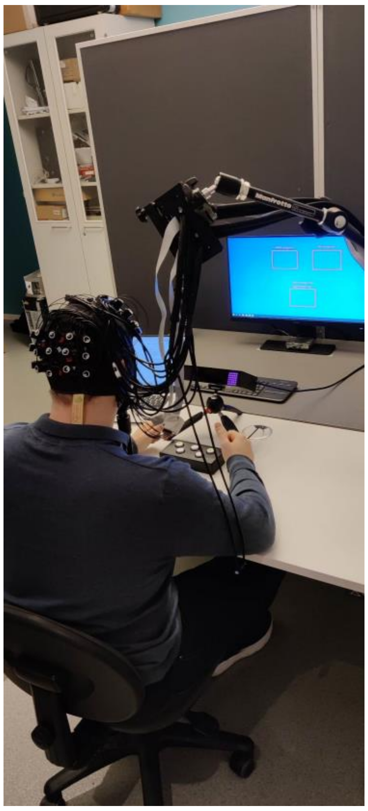
14) If fNIRS cap need to be changed,
- ☐ One researcher starts recapping (takes 15.min)
- ☐ The other researcher adjusts headstand and chair to match participant
- ☐ Participant can go out of the testroom and chill om their phone
- ☐ **Remember to ensure that the cables are organized and arranged in a neat and orderly manner [see phot of fNIRS cap bellow].**

15) Apply fNIRS-cap
- ☐ Remember to adjust the arm holding the fNIRS cap
- ☐ Let the participant attach the strap below the chin, should not be too tight/loose. <you should have room for a finger>
- ☐ Distance between channel 12 and 13 (3cm) should be the same as distance between 13 and middle of the eyebrows
- ☐ "Does this feel ok?"
- ☐ If the plastic clips breaks (the plastic holding the probes together):
  - ☐ Remove fNIRS cap
  - ☐ Replace it.

  Note: Spare parts are in bag [see photo]

- ☐ Arm should be above and close to the head [see picture right]
- ☐ Ensure that the cables are organized and arranged in a neat and orderly manner [see photo left]





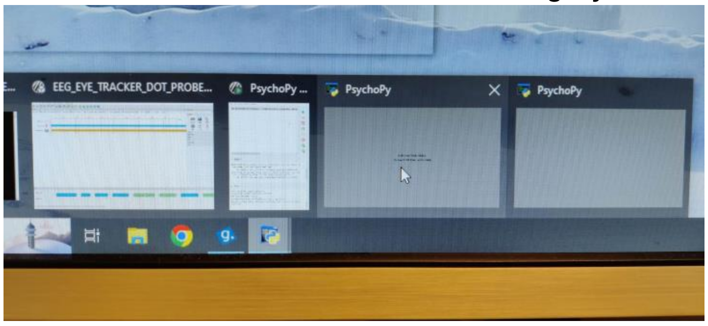
16) Calibration
- ☐ Drag the Nirstar program over to the participant screen.
- ☐ NB: Turn off the light before every calibration
- ☐ Click the **calibrate** button in the NIRStar program

  Note: The participant need to stay completely still during calibration
- ☐ All sections should be **green**, if problem, remove hair, twist on the detectors/sources to get better contact or use gel if it does not improve

17) Put on shower cap
- ☐ Run another calibration
- ☐ If it does not improve after several attempts, note down acceptable (yellow or red) channels in the experimental log
- ☐ Note down any error-messages in experimental log
- ☐ **Remember to drag the Nirstar program over to the experimenter screen.**

PREPARATIONS FOR EYE TRACKER

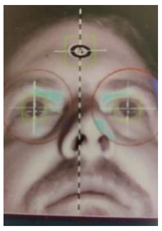
18) Start Psychopy program:
- ☐ Go to desktop → Open program in desktop with name “FNIRS_psychopy” (Its in the red square under fNIRS programs)
- ☐ Click run experiment in Psychopy
- ☐ Type in [ParticipantID]
- ☐ Make sure Psychopy is selected
    - ☐ Hover over psychopy icon at the taskbar.
    - ☐ Click on the grey window **with text** [see photo] (If no text in the window, choose the left-most grey window).



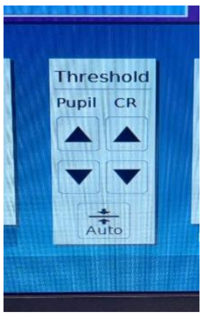
- ☐ Go to the participant screen.
    - ☐ Check that the participants screen is grey with the text: “Ready for eye-tracking calibration, press enter two times to continue”.
    - ☐ Use the wireless keyboard.
        1. Press ‘enter’ two times to display eye-tracking camera.
        2. Use left and right arrow to change view to show the whole face.
           Hint: other navigation keys are displayed at the top left corner.

19) Adjusting the eye tracker:
- ☐ Ask participant to lean into the headstand.
- ☐ If headstand not yet adjusted:
    - ☐ Ask participant to adjust chair so it is comfortable.
    - ☐ Adjust headstand so it is comfortable for participant.
- ☐ If they have glasses, make sure they are tucked up as far as possible.
    - ☐ Note: If the glasses cause to much problem, remove them, and note it in experiment log
- ☐ Make sure both pupils are visible on the display.
    - ☐ Turn red button to adjust eye-tracker

- ☐ Drag the red circles over each eye [see left photo]
- ☐ Use the wheel under the eye-tracker (to the left) to adjust the sharpness:
    - ☐ Use left and right arrow to display each eye.
    - ☐ Adjust until you get as sharp picture as possible.
    - ☐ Ask participant to look at four corners.
    - ☐ The teal-dot part should have as little white as possible [see left on photo bellow] (Do this with each eye in focus in the display)

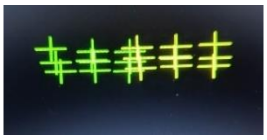


20) Pre-Calibrating the eye tracker (done on the host-pc/laptop):
- ☐ Find the perfect threshold --> in the grey rectangle --> click Auto [See photo]
- ☐ Threshold CR should be between 215 to 240
- ☐ Pupil threshold should be between 60 to 140
- ☐ If thresholds are too high:
    - ☐ change illumination level to 75%
    - ☐ Try go through step 18 again.
    - ☐ If values still not in threshold → note down in experiment log.
- ☐ To check if that threshold is good:
    - ☐ Participant should look to the four corners of the screen
    - ☐ The cross with the teal dot in the middle should keep pointing to the teal dot that is closest to the pupil [Middle and right on photo above illustrates what to avoid, Left shows how it should look]



21) Calibrating and validating the eye-tracker (done on the host-pc/laptop):
- ☐ "We will now run a quick calibration, please look at the dot at all times. Stare at the dot until it disappears. **Do it slowly**".
- ☐ Press C to initiate calibration mode.
- ☐ Press Space to start calibration.
- ☐ The figures/crosses should be aligned [see photo].
- ☐ If calibration failed
    - ☐ Click restart and run calibration again
      Note: The reason it failed could be that

      participant



22) Validate the calibration
- ☐ Press V for validating the calibration.
- ☐ Press space to start validation.
- ☐ Both eyes should have a GOOD validation (Is written in the bottom left corner)
    - ☐ If not, run calibration and validation again:
        1. Click abort and then 'c' to restart calibration.

2. If does not work after several times, note values in experiment log.

- ☐ Press ‘enter’ when validation is finished.

23)

24) When calibration is done

- ☐ Make sure Psychopy is selected
  - ☐ Hover over psychopy icon at the taskbar (same as in section 17). **(If no text in the window, choose the left-most grey window).**
  - ☐ Click on the grey window with eye-tracker displayed.
- ☐ Go to participant screen and use the wireless keyboard.
- ☐ Press ‘O’ , **wait 3 seconds** and then ‘enter’.
  The participant screen should then turn black with the text: “From a scale to 0-9, how tired are you?” **Note: If the screen just stays black, exit psychopy and start from section 17) again.**
  - ☐ Ask the participant and push the number they say.
- ☐ The text “Researcher will start data collection now …. “ should then appear.
  Note: If the

START TASKS AND RECORDING (1/2)

25) Double check that triggers are sent
   - ☐ In NIRStar, go to Configure Hardware --> Data streaming --> Receive triggers(LSL) --> Test connection (should be ok).

26) Read these instructions:

We will now begin the task on the computer. This task takes about 10 minutes. There will be instructions on the screen, but here is a short explanation:

This is a reaction test, in each trial of the experiment will start with a small ‘+’ (plus sign) in the center of the screen. At all times keep your eyes fixated on the plus sign. After an initial period of fixation two stimuli will be briefly presented: one on each side of the screen. After these stimuli disappear, a small dot will appear either on the left or on the right side of the screen.

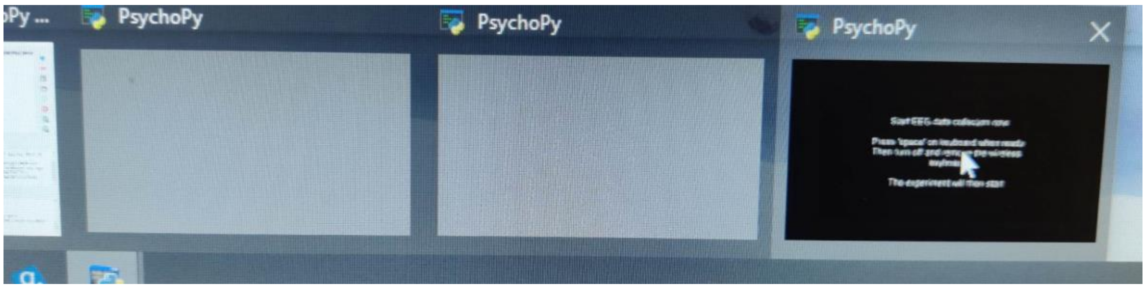
Your task is to locate this dot: left or right. Use your left index finger on the “L” button on the keyboard to indicate leftsided target dots. Use your right index finger on the “R” button on the keyboard to indicate right-sided target dots.

IT IS IMPORTANT THAT YOU RESPOND AS QUICKLY AS POSSIBLE. AS SOON AS YOU LOCATE THE DOT MAKE A RESPONSE.

You can take a small break after this test if you like. One of us will stay in the room, just let us know when you are done.

DO YOU HAVE ANY QUESTIONS?

27) Start the data collection:
- ☐ Click the **record** button in NIRstar
    - ☐ NB: Make sure to have NIRstar signal graph in background
- ☐ Click on the psychopy screen with black background and white text (If participant do not get the keyboard to work, this is because you have not clicked on the psychopy window) [See picture]



Start EEG-data collection now

Press 'space' on keyboard when ready
Then turn off and remove the wireless
keyboard

The experiment will then start

- ☐ When ready, press ‘space’ on wireless keyboard.
- ☐ Turn off the wireless keyboard and put it out of sight.
- ☐ The text “The task goes as follows…” should appear

28) One researcher must stay in the room to check:
- ☐ No problem with the data collection (keep an eye on the graph)
- ☐ Triggers are being sent (vertical dotted lines appearing on the graph)
    - ☐ No vertical lines: Cancel and call Sven (Phone: 99258430)
- ☐ Know when the participant is done with the task.
- ☐ **NB**: Be completely silent during the experiment and make sure phone is out of the room

29) **The participant can now start the experiment.**

WHEN PARTICIPANT IS DONE WITH TASKS (1/2)

30) Stop recordings
- ☐ On eyetracker (should be done automatically [screen should look like photo in section 5])
- ☐ On fNIRS (Nirstar → Stop button on right side)
    - ☐ Click ‘ok’ on message saying “Could not export coordinates ..”

- ☐ Shut down NirScout (Green button)
- ☐ Shut down “Cyber power” (black box under desk)
- ☐ Close all programs.

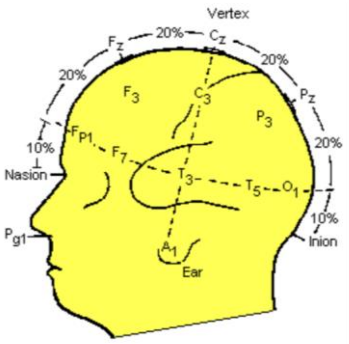
31) Say to participant
- ☐ “Thank you, first part out of two is done. Now we are going to take off the equipment”

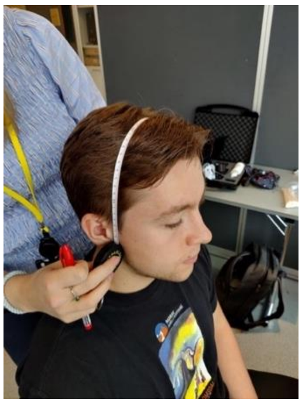
32) Take cap off participant
- ☐ Do you want a break, or do you need to use the restroom?

33) fNIRS equipment
- ☐ Place fNIRS cap on Robscar
- ☐ Lower arm holding fNIRS equipment

34) Turn on wireless keyboard and place on participant table.

PREPARATION FOR EEG





Lower fNIRS arm, and place it a nice place

35) Find mid-point (Cz) of the head
- ☐ Measure distance between pre-auricular points (ear-ear)
    - ☐ divide by 2 to find middle
    - ☐ Mark with lipliner
- ☐ Measure distance between nasion and inion
    - ☐ divide by 2 to find middle
    - ☐ Mark with lipliner

36) Apply EEG-cap
- ☐ Put Cz (nr. 16 on cap) on the middle of the head (--where you have marked with the lipliner.) Keep ears free from hair.
- ☐ Let the participant attach the strap below the chin, should not be too tight/loose.

37) Connect reference to earlobe
- ☐ Apply gel to silver plate before

38) Connect cap to amplifier and turn on amplifier
- ☐ *Hold for 5 seconds or more*

- ☐ When the receiver and the amplifier is connected, the blinking turns from a fast blinking to a slower blinking.
- ☐ If it is blinking fast, it means it is not connected with the basestation
- ☐ In data acquisition mode, the led is permanently on.

39) Set up EEG software
- ☐ Open g.tec suite
    - ☐ Go to: Applications --> g.recorder
- ☐ In g.recorder, choose setup
  File --> Load setup... --> choose DOT_PROBE_SETUP
- ☐ add the EEG equipment
  Settings --> select hardware --> g.Nautilus --> click arrow to right --> Click ok
- ☐ *Select montage*
  File --> Load electrode montage --> select the file: gNautulus_32ch

40) Start impedance check:
- ☐ Go to Tools --> Impedance measurements.
- ☐ Change topography to 2D topographic.
- ☐ Under display option --> display mode --> choose Impedance values
  Click Start
- ☐ Drag the impedance measurement window over to the participant screen.

41) Improve impedance:
- ☐ **Start with the ground (GND) electrode before you do the other electrodes**.
- ☐ Remove hair carefully with the tip of the syringe (butterfly!!)
- ☐ Fill the electrode with gel so you can see the gel coming up from the hole.
- ☐ All electrodes should be green and have an impedance value below 25-30 kΩ before starting experiment.
    - ☐ Experience a lot of problem with getting a good value? note it down in research log --> procced with the experiment
    - ☐ BE CAREFUL OF BRIDGING: The gel applied on two nearby electrodes may come in contact with each other, this Is called bridging
        1. How to notice:
            a. Nearby electrodes have the same value
            b. Nearby electrodes change value at the same time

Note: If all electrodes turn black → Get better connection on ground electrode

42) When impedance measurement is done:

- ☐ Click stop.
- ☐ Drag it back to the experimenter screen.
- ☐ Close impedance check

43) Start dataviewing
- ☐ TURN OFF FNIRS AND REMOVE ALL PHONES/ELECTRONIC OUTSIDE OF THE ROOM!
- ☐ Press the dataviewing button (play button)
- ☐ Press autoscale each channel to its individual min/max amplitudes

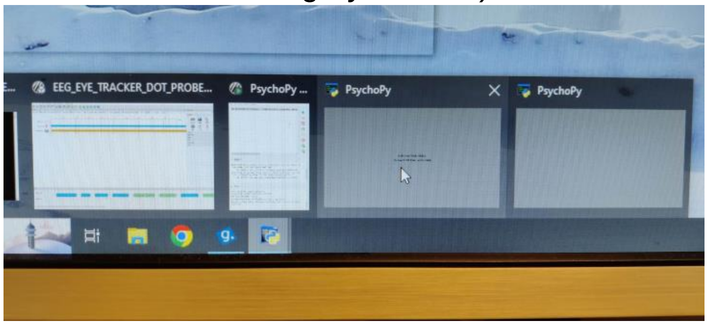
If you have time, pull the screen over to the participant so they can look. They usually think this is cool

44) Ask participant to:
- ☐ Sit calm and look at screen (don't talk and don't move)
- ☐ Blink 3 times --> look for frontal electrodes activation (Upper one at the screen)
- ☐ Bite for 2 sec --> electrodes near middle should show more activity
- ☐ Close eyes --> alpha activity on electrodes at the back of the head (more activation on bottom electrodes)

45) Pause dataviewing (if relevant, pull the screen back to the experimenter pc)

PREPARATIONS FOR EYE TRACKER

46) Start Psychopy program:
- ☐ Go to desktop → Open program in desktop with name "EEG_Psychopy" (Its in the red square under EEG programs)
- ☐ Click run experiment in Psychopy
- ☐ Type in [ParticipantID]
- ☐ Hover over psychopy icon at the taskbar.
- ☐ Click on the grey window with text [see photo] (If no text in the window, choose the left-most grey window).

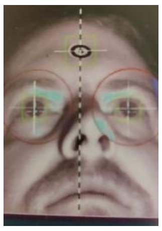


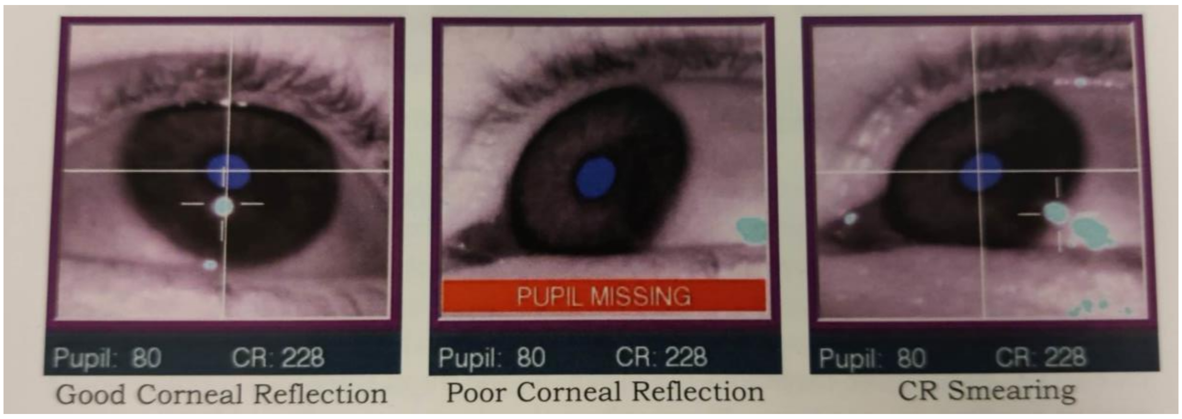
- ☐ Go to the participant screen.

- ☐ Check that the participants screen is grey with the text: “Ready for eye-tracking calibration, press enter two times to continue”.
- ☐ Use the wireless keyboard.
    - ☐ Press ‘enter’ two times to display eye-tracking camera.
    - ☐ Use left and right arrow to change view to show the whole face.
    - ☐ Other navigation keys are displayed at the top left corner.

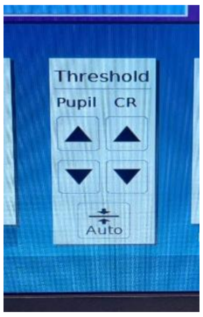
47) Adjusting the eye tracker:

- ☐ Ask participant to lean into the headstand.
- ☐ If they have glasses, make sure they are tucked up as far as possible.
    - ☐ Note: If the glasses cause to much problem, remove them, and note it in experiment log
- ☐ Make sure both pupils are visible on the display.
    - ☐ Turn red button to adjust eye-tracker.
- ☐ Drag the red circles over each eye [see left photo]
- ☐ Use the wheel under the eye-tracker (to the left) to adjust the sharpness:
    - ☐ Use left and right arrow to display each eye.
    - ☐ Adjust until you get as sharp picture as possible.
    - ☐ Ask participant to look at four corners.
    - ☐ The teal-dot part should have as little white as possible [see left on photo bellow] (Do this with each eye in focus in the display)





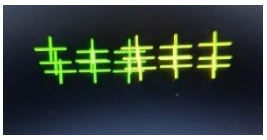
48) Pre-Calibrating the eye tracker (done on the host-pc/laptop):



- ☐ Find the perfect threshold --> in the grey rectangle --> click Auto [See photo]
- ☐ Threshold CR should be between 215 to 240
- ☐ Pupil threshold should be between 60 to 140
- ☐ If thresholds are too high:
    - ☐ change illumination level to 75%
    - ☐ If values still not in threshold → note down in experiment log.
- ☐ To check if that threshold is good:
    - ☐ Participant should look to the four corners of the screen
    - ☐ The cross with the teal dot in the middle should keep pointing to the teal dot that is closest to the pupil [Middle and right on photo above illustrates what to avoid, Left shows how it should look]

49)Calibrating and validating the eye-tracker (done on the host-pc/laptop):

- ☐ "We will now run a quick calibration, please look at the dot at all times. Stare at the dot until it disappears".
- ☐ Press C to initiate calibration mode.
- ☐ Press Space to start calibration.
- ☐ The figures/crosses should be aligned [see photo].



50)Validate the calibration:

- ☐ Press V for validating the calibration.
- ☐ Press space to start validation.
- ☐ Both eyes should have a GOOD validation (Is written in the bottom left corner)
    - ☐ If not, run calibration and validation again:
        1. Click abort and then 'c' restart calibration
        2. If does not work after several times, note values in experiment log.
- ☐ Press 'enter' when validation is finished.

51)When calibration is done

- ☐ Hover over psychopy icon at the taskbar (same as in section 17).
- ☐ Click on the grey window with eye-tracker displayed.
- ☐ Go to participant screen and use the wireless keyboard.
- ☐ Press 'O', **wait 3 seconds** and then 'enter'.
- ☐ The participant screen should then turn black with the text: "From a scale to 0-9, how tired are you?" **Note: If the screen just stays black, exit psychopy and start from section 17) again.**
    - ☐ Ask the participant and push the number they say.
- ☐ The text "Researcher will start data collection now …. " should then appear.

START TASKS AND RECORDING (2/2)

52) Read these instructions:

We will now begin the final out of two tasks on the computer. This tasks takes about 20 minutes. There will be instructions on the screen, but here is a short explanation:

The first task requires you to sit still and relax while looking at a fixation cross ('+' sign). This task will take 2,5minutes.

The second next task is the same, but with closed eyes. You will hear a beep when it is done.

The third is the same as the one you did before with the two images, the reaction test. In the reaction test, IT IS IMPORTANT THAT YOU RESPOND AS QUICKLY AS POSSIBLE. AS SOON AS YOU LOCATE THE DOT MAKE A RESPONSE.

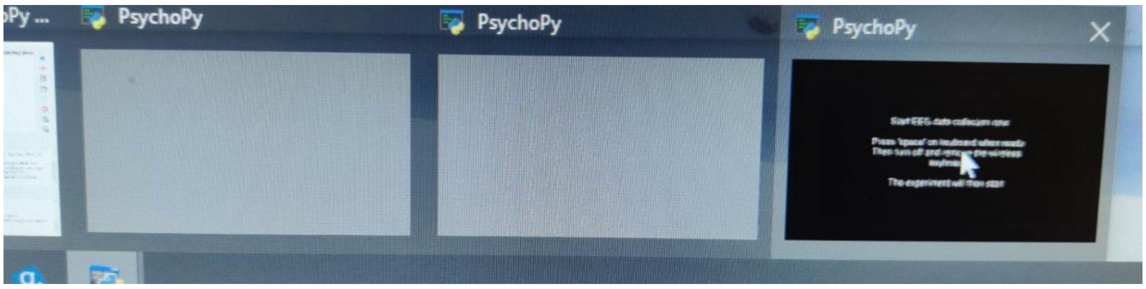
The fourth is a number of short visual tasks which consists of following you a dot with your gaze, looking at pictures and 'find Waldo' tasks.

DO YOU HAVE ANY QUESTIONS?

53) Start the data collection:

- ☐ Click the **record** button in g.recorder
    - ☐ Choose path for the patticipant folder
        1. Skrivebord→Data → Participant ID
    - ☐ Filename "ParticipantID_EEG_"
    - ☐ NB: Make sure to have g.recorder signal graph in background
- ☐ Turn volume to 40%

- ☐ Click on the psychopy screen with black background and white text (If participant do not get the keyboard to work, this is because you have not clicked on the psychopy window) [See picture]



Start EEG data collection now

Press 'space' on keyboard when ready
Then turn off and remove the wireless keyboard

The experiment will then start

- ☐ When ready, press ‘space’ on wireless keyboard.
- ☐ Turn off the keyboard and put it out of sight.
- ☐ The text “You are now going to go through a number of tasks…” should appear

54) One researcher leaves the room the other must stay in the room to check:

- ☐ **Don’t click anything on the experimenter PC, this will disrupt the program**!
- ☐ No problem with the data collection (keep an eye on the graph)
- ☐ Triggers are being sent (vertical dotted lines appearing on the graph)
  - ☐ No vertical lines: Cancel and call Sven (Phone: 99258430)
- ☐ Know when the participant is done with the task.
- ☐ **NB**: Be completely silent during the experiment and no phone inside the lab

55) **When one researcher has left, let the participant know they can start the experiment.**

WHEN PARTICIPANT IS DONE WITH TASKS (2/2)

56) STOP EEG recording.
57) Close all programs (including psychopy)
58) Help participant take off the cap:
    - ☐ Reference electrode on the earlobe
    - ☐ Disconnect cap from amplifier.
    - ☐ Turn off amplifier.
    - ☐ Take off cap gently
59) Give participant giftcard
    - ☐ Ask participant to note down giftcard number and sign paper.
    - ☐ get giftcard-paper from participant and store it in the pink folder.
60) Give participant:
    - ☐ Towel
    - ☐ Shampoo
    - ☐ Conditioner
    - ☐ And guide to shower (in basement downstairs)
61) Say thank you and goodbye.

AFTER EXPERIMENT

STORE DATA TO TSD (TAKES ~20 MIN TO UPLOAD DATA, SO DO THIS BEFORE COMPLETING OTHER TASKS)

62) Copy and drag data into the participant folder
- ☐ Go to Eye_FNIRS → "[ParticipantID]_NIR_time
  - ☐ Copy from file with name "**[ParticipantID]_NIR_time.EDF** "(EyeLink Data File).
- ☐ Go to Eye_EEG → "[ParticipantID]_EEG_time
  - ☐ Copy file with name **"[ParticipantID]_EEG_time.EDF "(EyeLink** Data File).
- ☐ Go to folder with name "EEG_Data_Psychopy"
  - ☐ Copy excel file named "**[ParticipantID]_EEG_DP_time"**
- ☐ Go to folder with name "FNIRS_Data_Psychopy"
  - ☐ Copy excel file named **"[ParticipantID]_FNIRS_DP_time"**

63) **Make sure all this data is inside the participant folder (7 in total) :**
1. Folder with name **"[ParticipantID]_fNIRS "**
2. **[ParticipantID]_EEG_date_time.hdf5"**
3. **[ParticipantID]_NIR_time.EDF**
4. **[ParticipantID]_EEG_time.EDF**
5. **[ParticipantID]_EEG_DP_time.csv**
6. **[ParticipantID]_FNIRS_DP_time.csv**
7. **Log_ParticipantID.docx**

64) Create ZIP file
- ☐ Go to desktop → data
- ☐ Mark folder with name «Participant ID» → right click → choose "Send til» → choose «komprimmert zippet mappe»

65) Go to TSD to import files
- ☐ Go to desktop → Click the TSD icon (under "data storage and experimental log")
- ☐ If it does not work, go to this link in chrome :
  - ☐ https://data.tsd.usit.no/i/30b8a851-1476-440c-b20d-916a03ae5a81
- ☐ Click "login"

- ☐ Click “Import Files” and add the ZIP folder in desktop → data → “[ParticipantID].z”
- ☐ Press “Import” and wait for the data to be uploaded

66) Put EEG amplifier to charging:
- ☐ Turn device off. Remove from housing to ensure direct connection. Put on charging plate The LED turns on when charging, with a bit different color. Batteri can measure 10 hours conitnius recording. 2-2,5 hours to recharge again. Can be on charging plate.

67) Cleaning the EEG cap:
- ☐ Attach the EEG cap to the neckband (always have around your neck when you are cleaning!)
- ☐ Get the washing bowl, toothbrush and small brushes
- ☐ Fill bowl with lukewarm water/ or use the sink in the toilet (if not too busy)
- ☐ Use the toothbrush and small brushes to clean the EEG
    - ☐ **START TO CLEAN REF and GND**
    - ☐ Clean both sides of each electrode thoroughly (also in the holes)
- ☐ Can use hair dryer If you need to dry quickly, don’t be too close. Half a meter.
- ☐ Clean the top of gel flask
- ☐ Clean syringes
- ☐ **Make sure NO GEL is left, this will dry out and ruin the electrodes**

68) EyeT
- ☐ Clean stativ with wipes

If relevant: collect towel from participant in shower in basement

Fill out giftcard file

Teams channel “datacollection 23” --> giftcard

CHECKLIST AFTER EXPERIMENT

- Experiment log is filled out
- **All equipment is turned off**
  - **Computers**
  - **fNIRS**
  - **EEG amplifier**
  - **Eye-tracker**
- **Equipment is cleaned**
  - **EEG cap (all electrodes including reference)**
  - **Syringes**
  - **Top of gel**
- **All equipment is placed inside the grey drawer and locked**
- Amplifier for EEG is charging
- Cabins are clean
  - Clean table
  - No food/other stuff
- Make sure the file has been stored correctly. (Check participant folder).

It should contain:

  - Folder with name **"[ParticipantID]_fNIRS "**
  - [ParticipantID]_EEG_date_time.hdf5"
  - [ParticipantID]_NIR_time.EDF
  - [ParticipantID]_EEG_time.EDF
  - [ParticipantID]_EEG_DP_time.csv
  - [ParticipantID]_FNIRS_DP_time.csv

  Log_ParticipantID.docx

- Participant folder is imported into TSD

A.2 Research Paper

Detection of biomarkers from dot-probe task using functional near-infrared spectroscopy

Sven I. Ougendal,[a] Rune Jonassen,[b] Peyman Mirtaheri,[a,*]
[a]Oslomet-Oslo Metropolitan University, Faculty of Technology, Art and Design, Department of Mechanical, Electronic and Chemical Engineering, Pilestredet 35, Oslo, Norway, 0166
[b]Oslomet-Oslo Metropolitan University, Faculty of Health Sciences, Department of Nursing and Health Promotion, Pilestredet 32, Oslo, Norway, 0166

Abstract. Over the past decade, there has been a notable increase of 13% in mental health issues. Approximately one in five children and adolescents globally suffer from a mental health problem. Cognitive biases, such as attentional bias (AB), may contribute to the onset and persistence of mental health disorders. AB can be defined as the tendency to selectively attend to or focus on certain stimuli while ignoring others, where someone with a negative AB have a disproportional attention to negative stimuli. Understanding the underlying neural processes involved in AB and identifying reliable biomarkers may be important in developing successful interventions for mental health disorders. The dot-probe task is one of the most widely experiments used to detect AB, where it uses reaction time as a measurement of AB. However, traditional methods for assessing AB using RT indices have demonstrated weak internal consistency and limited test-retest dependability. This suggest using other measurements in combination with the dot-probe task could help enhance our understanding of AB. A large-scale experiment on 60 healthy women between 18-30 was conducted, using functional near-infrared spectroscopy (fNIRS) in combination with the dot-probe task. A multi-subject GLM analysis was done on the fNIRS data, which focused on detecting significant variations in oxyhemoglobin (HbO) and deoxyhemoglobin (HbR) concentrations between congruent (dot behind the face with the most negative emotion) and incongruent dot-probe trials in the prefrontal cortex (PFC) and visual cortex regions of the brain. The results of the multi-subject general linear model (GLM) analysis revealed two significant findings: (1) a higher HbR concentration in the right ventral medial PFC when the dot is located behind the fearful face compared to when it is behind the neutral face, and (2) a higher HbR concentration in the right dorsal PFC when the dot is positioned behind a face expressing emotion compared to when it is behind a neutral face. These findings highlight the potential of using fNIRS to study AB.

Keywords: functional near-infrared spectroscopy (fNIRS), Attentional bias (AB), dot-probe task, Prefrontal cortex (PFC).

***Peyman Mirtaheri**, E-mail: peymanm@oslomet.no

Introduction

Over the past decade, there has been a notable increase of 13% in mental health issues. Approximately one in five children and adolescents globally suffer from a mental health problem, with suicide ranking as the second most common cause of death for individuals aged 15-29. Depression and anxiety have a combined economic impact of US$ 1 trillion annually (WHO, 2023).

Cognitive biases, such as AB, may contribute to the onset and persistence of mental health disorders. AB can be defined as the tendency to selectively attend to or focus on certain stimuli while ignoring others, where someone with a negative AB pays disproportional attention to negative stimuli. For instance, individuals with depression often focus disproportionately on dysphoric stimuli, while those with anxiety are prone to being easily side-tracked by potential threats (Barry et al., 2015; Disner et al., 2011).

Understanding the underlying neural processes involved in AB and identifying reliable biomarkers may be important in developing successful interventions for mental health disorders. By incorporating cost-efficient and portable cognitive and neurofunctional measures of AB alongside conventional mental health evaluations, we can potentially enhance the accuracy of individualized treatment response predictions (Barry et al., 2015).

Numerous computerized experimental tasks have been utilized by researchers to investigate AB. The tasks employed include spatial cueing-, visual search-, Stroop-, and dot-probe tasks (Chew, 2015). Among these, the dot-probe task is regarded as the "gold standard" by some experts, as it remains the most widely employed technique in AB research (Kappenman et al., 2014; Torrence & Troup, 2018). The task employs reaction time (RT) as an indicator of AB. However, traditional methods for assessing AB using RT indices have demonstrated weak internal consistency and limited test-retest dependability, as reported in various studies (Brown et al., 2014; Schmukle, 2005; Staugaard, 2009). Researchers have also investigated other RT-based metrics, such as those based on variability, but these alternatives have proven to be inconsistent as well (Carlson & Fang, 2020; Naim et al., 2015; Price et al., 2015). Incorporating supplementary assessment methods could offer additional insights into AB (Carlson, 2021; Carlson & Fang, 2020).

To better understand AB, measurement of different brain activities has been used in combination with the dot-probe task, mainly electrical and perfusion. Electrical activity refers to the changes in electrical potentials generated by neurons, which can be measured using Electroencephalography (EEG). Perfusion changes refer to the changes in blood flow and oxygenation levels in the brain, which can be measured using functional magnetic resonance imaging (fMRI) and fNIRS (Carlson & Fang, 2020; Price et al., 2014; Torrence, 2015). Electrical and perfusion changes are linked together through neurovascular coupling, where an increase in neuronal activity drives changes in blood flow and oxygenation to meet the demands of active brain regions, resulting in concentration changes in HbO and HbR (Khan et al., 2021).

Although fNIRS is relatively new and unexplored in AB research, it presents several advantages over fMRI, including measurements of both HbO and HbR, superior temporal resolution, substantially cheaper equipment, and the capacity to assess changes in cortical regions in more natural settings compared to fMRI machines (Ehlis et al., 2014). fNIRS uses at least two wavelengths of light to measure concentration changes in HbO and HbR. The procedure involves sending near-infrared light into the brain tissue through the skull, where Hb absorbs it in the blood. Depending on the oxygenation status of the Hb, different amounts of light are absorbed by it. fNIRS can determine which regions of the brain are active during a specific task or at rest by observing these changes (Quaresima & Ferrari, 2019).

While research on AB using fNIRS is limited, one investigation observed alterations in HbO concentrations in both the medial PFC and bilateral PFC during congruent and incongruent trials in a dot-probe task involving fearful faces (Torrence, 2015). This finding suggests that fNIRS might be a valuable tool for examining AB. In addition, a recent systematic review on AB recommended a multimodal approach to measuring AB to improve the reliability and validity of assessments (Carlson, 2021). By combining multiple measures, it may be possible to gain a more comprehensive understanding of AB and its underlying neural processes.

Methods

Participants

Sixty healthy adult female participants, selected from a random population, took part in the experiment. All participants were within the age range of 18 to 30 years. As part of our inclusion criteria, all participants were required to be in good health and not on any medication. Normal vision or vision corrected to normal was a prerequisite for participation. The exclusion criteria were established to eliminate potential confounding variables that could influence the study outcomes. These included severe psychopathology, brain injury, neurological disorder, or any medication use. We aimed to ensure that our sample was as representative as possible of the general population, excluding any factors that could potentially skew the results.

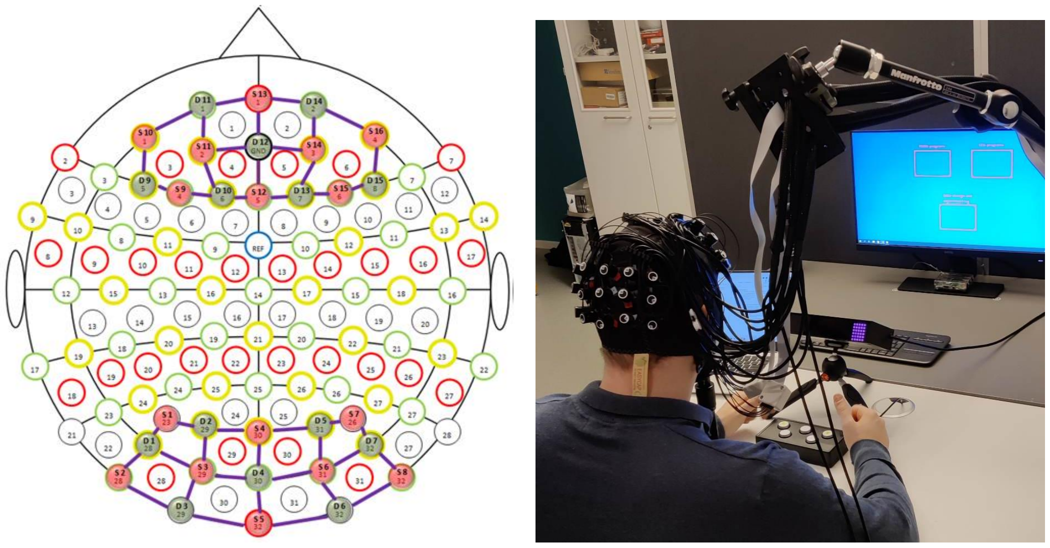
Recruitment and ethics

To recruit participants for the experiment, we used multiple methods such as personal networks, social media, and Instagram ads. Participants were directed to an online survey portal called "Nettskjema" by scanning a QR code. This service, provided by the University of Oslo (UiO), allowed for secure data storage with restricted access limited to researchers and students related to the project. Upon meeting eligibility criteria, participants were contacted by the researcher to schedule an appropriate time for testing. A consent form was sent to the participants, and they were asked to return a signed form before the scheduled experiment.

We applied for ethical approval from the Regional Committee for Medical and Health Research Ethics (REK) and stated that only "healthy" participants would be included. To protect privacy, each participant was assigned a unique ID number, and personal information was stored in a separate paper document under lock and key. All data will be initially stored on a local device and then uploaded to a cloud service called "Tjenester for sensitive data" (TSD), designed specifically for sensitive data storage. Consent forms was stored separately and deleted from the researcher's email inbox.

Instrumentation

The fNIRS equipment used is the **NIRScout** system, manufactured by NIRx (Berlin, Germany). This equipment utilizes continuous-wave technology to measure Hb levels. This system applies two wavelengths of 760nm and 850nm with optical fibres. Our experimental setup employs a 42-channel, 16x16 prefrontal and occipital cortex montage, adhering to the standard 10/20 arrangement. The sampling rate for our study is set to 3.91 Hz. The probes on the NIRScout are wired, we have therefore attached a cable holder to the table to alleviate the weight of the wires on the participant's head, ensuring their comfort. The **NIRStar 15-3** software, which is included with the NIRScout system, was used for the data-acquisition.



(a (b

Fig. 1: (a) fNIRS montage setup and (b) experimental setup.

Experimental setup and instruction

The experiment was conducted in a carefully controlled setting. We maintained consistent lighting conditions for all participants and utilized a shower cap on the fNIRS device to shield it from outside light. Although we minimized external noise as much as possible, we were unable to eliminate the low hum of a nearby tram. However, we anticipated that this faint noise would not significantly affect the results. A 1080p monitor, measuring 53x30cm, was used for the experiment. Participants were seated comfortably in a chair with their heads resting on a headstand, positioned approximately 115cm away from the screen. Before the experiment began, we provided them with a clear verbal explanation of the task they would be undertaking. This was supplemented with written instructions displayed prior to the start of the experiment. To record participants' responses, we employed a response box created by SR-research. The buttons were labelled 'L' for left and 'R' for right. Participants were instructed to rest their left index finger on the left button and their right index finger on the right button.

Experiment design

Each trial starts with a white fixation cross displayed at the center of the screen for a randomized duration of either 750ms or 1250ms. The fixation cross remains visible until the dot disappears. Two faces (happy/neutral, fearful/neutral, or neutral/neutral pairs) are presented for 1200ms, spanning 5° × 7° of the visual field, with roughly 14° separating the innermost borders of the facial stimuli. Immediately after the faces vanish, a white dot appears behind either the left or right image, centered within the picture. Trials are categorized as congruent if the dot is behind the face with the most negative emotion (e.g., behind the neutral face in happy/neutral pairs), and incongruent otherwise.

Participants are required to indicate the dot's location by pressing the 'left' or 'right' button on the response box with their index finger. Following their response, a black screen is displayed for 1500ms, marking the end of the trial.

The participants completed twenty blocks, each consisting of four trials. Each condition is presented an equal number of times (four instances per condition): neutral/neutral, happy/neutral congruent, happy/neutral incongruent, fearful/neutral congruent, and fearful/neutral incongruent. Each block maintains an equal ratio of male to female faces, dot locations (left/right), and time jitter (750ms/1250ms). Additionally, a unique face is displayed in every trial, ensuring no repetitions. The participant goes through a total of 80 different trials.

After completing each block, participants are shown their best reaction time, followed by a 15-second rest period. The order of blocks and trials within them is counterbalanced across participants.

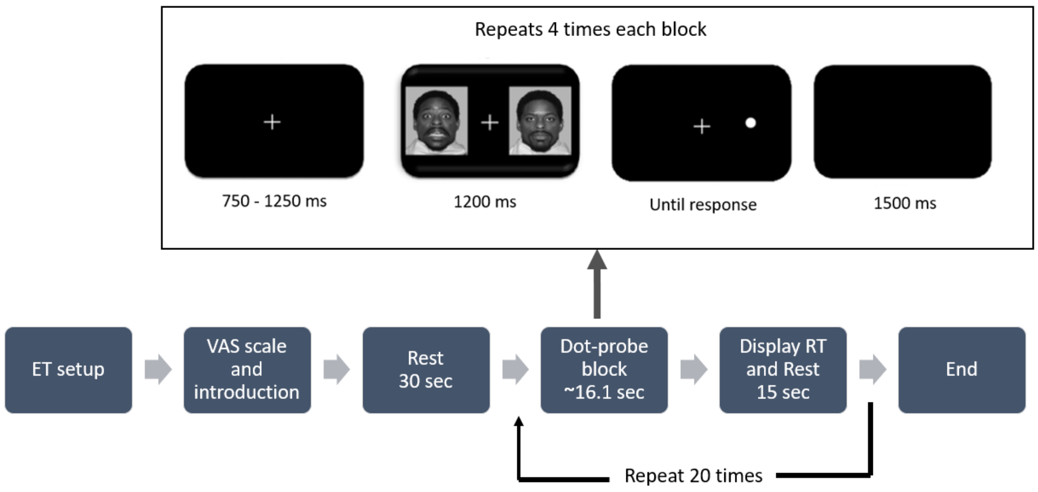

Repeats 4 times each block
750 - 1250 ms
1200 ms
Until response
1500 ms
ET setup
VAS scale and introduction
Rest 30 sec
Dot-probe block ~16.1 sec
Display RT and Rest 15 sec
End
Repeat 20 times

Fig. 2: Illustration of the experiment paradigm.

Signal pre-processing

For fNIRS data pre-processing and analysis, we employed the **Satori software**, developed collaboratively by Brain Innovation and NIRx. Satori was used for diverse tasks such as removing discontinuities, spikes, and truncation of the data points before and after the first and last stimuli appeared, respectively.

Bad channels were identified using the criterion of the coefficient of variation (CV) of 7%. The coefficient of variation is equal to a hundred times the standard deviation divided by the mean value of the raw data measurements. A large value for CV is an indication of high noise.

Next, we converted the data. The steps involved include converting intensity time-series into attenuation shifts (optical density) and then into concentration changes of HbO and HbR.

Although Satori doesn't reveal its calculation method, it's typically done using the modified Beer-Lambert law (Delpy et al., 1988).

Determining event block duration is challenging due to varying reaction times across trials. Satori only allows setting a uniform duration for all events, making individual adjustments laborious with 20 different events for 60 participants. To resolve this, we calculated an average event duration using the mean reaction time of all participants, which turned out to be 16.1 seconds (with a standard deviation of 0.37 seconds). This uniform duration might influence the fNIRS analysis results as some blocks may include rest period data or exclude relevant data. However, the impact is likely minimal due to the low sampling rate of fNIRS technology, which limits potential data loss or inclusion within the 16.1-second block. Moreover, the small standard deviation of 0.37 seconds suggests that the fixed block duration is unlikely to significantly alter the fNIRS analysis results.

We then used Temporal Derivative Distribution Repair (TDDR) to correct motion artefacts like spikes and baseline shifts. This is a motion correction technique developed by (Fishburn et al., 2019).

We calculated our filter's cut-off frequencies following guidelines by (Pinti et al., 2019). We first determined the stimulation frequency range. Given that the stimulation block's duration varied, we found the minimum and maximum time using the mean plus or minus the standard deviation, resulting in values of 15.73 seconds and 16.47 seconds, respectively. With a rest time of 15 seconds, we estimated the stimulus frequency range to be approximately [0.0317, 0.0325] Hz. Then we identified frequencies to include and exclude, such as heartbeats (1 Hz), respiration (0.3 Hz), and Mayer waves (0.1 Hz). Considering these factors, we selected a frequency range of [0.01, 0.09] Hz.

We used a Gaussian smoothing low-pass filter (cut-off frequency of 0.09 Hz) to eliminate high-frequency noise, and a 2. Order Butterworth high-pass filter (cut-off frequency of 0.01 Hz) to remove low-frequency drifts. The Gaussian filter preserves more frequencies, which was recommended in the Satori manual for subsequent GLM analysis (Brain-Inovation, 2023).

We utilized the z-normalization technique, which involves mean-centring the signal in a channel, relating it to the standard deviation fluctuations. This helps in understanding the deviation of individual values or mean effects within standard statistical frameworks.

Analysis

We conducted an analysis using a multi-subject GLM approach in Satori, where we followed the Satori Multi-Subject GLM Guide (Lührs et al., 2022) and Satori user manual (Brain-Inovation, 2023). The GLM analysis assumes that the residuals (the noise in the data) are uncorrelated. However, fNIRS data often contain serial correlations due to trends or physiological noise. To improve the accuracy of the GLM, these correlations were removed. Satori uses a process called pre-whitening to remove these, more info about this can be found in the Satori user manual (Brain-Inovation, 2023).

We selected Separate Subject Analysis, which involves estimating subject-specific beta values for each subject and condition within the Multi-Study GLM list. This approach follows the classical methodology for calculating a Random Effects GLM, which allows for potential generalization of effects beyond the measured sample.

To address the multiple comparisons problem in our data analysis, we applied the False Discovery Rate (FDR) correction method by (Benjamini & Hochberg, 1995). This method controls the proportion of false positives among the significant results, rather than the overall number of false positives, making it a suitable choice for fNIRS data analysis (Lührs et al., 2022). By using the FDR correction, we aimed to identify truly significant channels while accounting for the multiple comparisons problem. The FDR method adapts to the amount of activity in the data and maintains a high sensitivity to detect true effects. As a result, we minimized the risk of false positives and ensured more accurate results in our fNIRS data analysis.

We proceeded to create several contrast maps. A contrast map is a statistical map that highlights brain regions with significant differences or relationships between conditions. The contrast map is then generated by applying the contrast to the beta weights across all channels, resulting in a statistical value, t-value, for each location. In GLM contrasts, the '>' symbol compares the effects of two conditions. A positive t-value, colored red in 3D and 2D views, indicates the left side has a stronger effect, while a negative t-value, colored blue, means the right side has a stronger effect.

Results

Our GLM analysis looked at 4 different contrast maps:

1. Happy/Neutral Congruent > Happy/Neutral Incongruent
   (Dot behind happy emotion vs. behind neutral emotion)

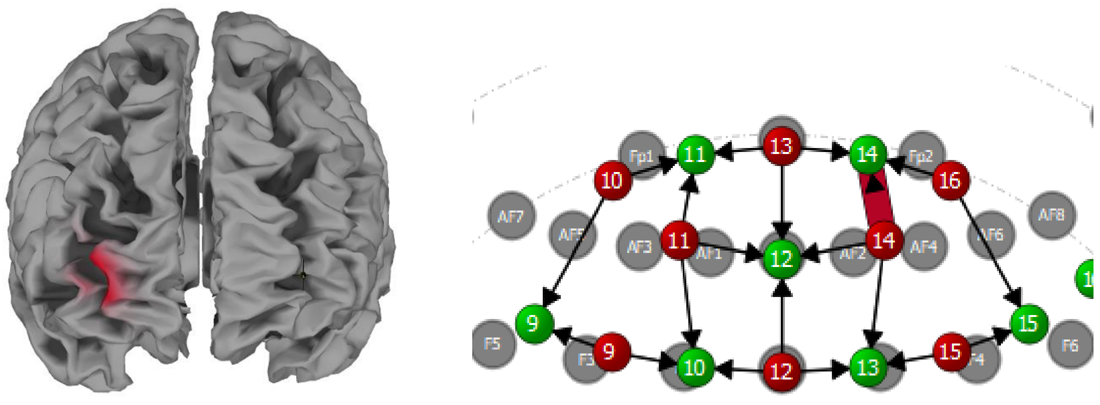
2. Fearful/Neutral Congruent > Fearful/Neutral Incongruent
   (Dot behind fearful emotion vs. behind neutral emotion)

3. Happy/Neutral Congruent + Fearful/Neutral Congruent >
   Happy/Neutral Incongruent + Fearful/Neutral Incongruent
   (Dot behind most negative emotion > Dot behind most positive emotion)

4. Happy/Neutral Incongruent + Fearful/Neutral Congruent >
   Happy/Neutral Congruent + Fearful/Neutral Incongruent
   (Dot behind emotional face > Dot behind neutral face)

We found a significant difference in contrast map number 2 and 4. Here are the result:

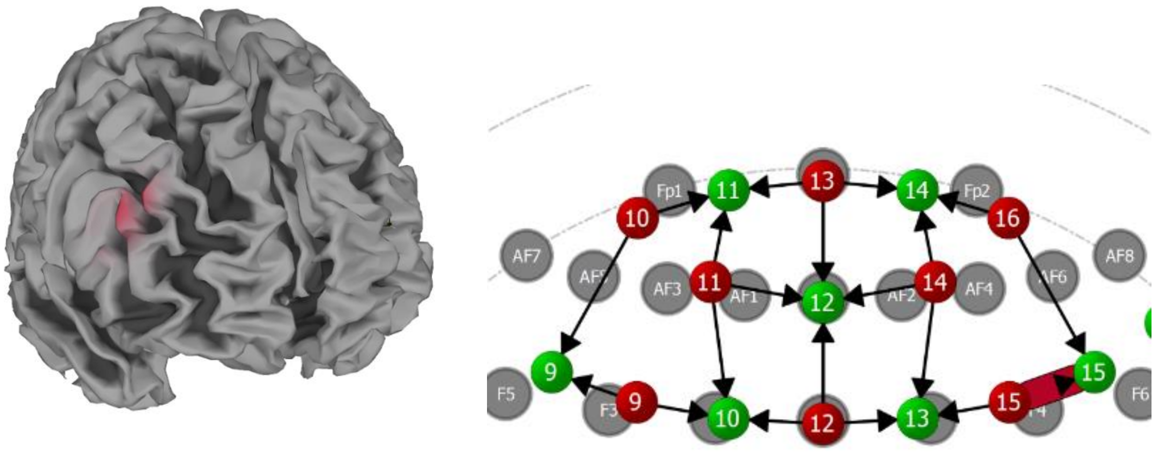
Contrast Map 2: This map identified a statistically significant difference in HbR concentration between two conditions: Fearful/Neutral Congruent trials > Fearful/Neutral Incongruent trials. The observed t-value of 2.544812 represents the magnitude of the difference between the conditions in terms of standard errors. The corresponding p-value of 0.01357, which is less than the commonly used threshold of 0.05, suggests that this difference is unlikely to have occurred by chance alone. In practical terms, this result implies that there is a higher HbR concentration in the right ventral medial PFC when the dot is located behind the fearful face compared to when it is behind the neutral face. See images below to see the contrast map in 3D and 2D view.



(a (b

Fig. 3: Illustration of significant differences in contras map 2: (a) 3D and (b) 2D.

Contrast Map 4: This map revealed a statistically significant difference in HbR concentration at channel 15-15 between two conditions: Happy/Neutral Incongruent + Fearful/Neutral Congruent > Happy/Neutral Congruent + Happy/Neutral Incongruent. With a t-value of 2.961729 and a p-value of 0.004403 (which is below the commonly used threshold of 0.05), the observed difference is unlikely to be due to chance alone. In practical terms, this result indicates a higher HbR concentration in the right dorsal PFC when the dot is positioned behind a face expressing emotion compared to when it is behind a neutral face. See images bellow to see the contrast map in 3D and 2D view.



(a (b

Fig. 4: Illustration of significant differences in contras map 4: (a) 3D and (b) 2D.

Discussion

The findings of the study are consistent with prior research suggesting that distinct neural activation patterns underpin AB, as observed through neuroimaging approaches such as EEG with event-related potentials (ERP) and fMRI (Britton et al., 2013; Carlson & Fang, 2020; Price et al., 2014; Torrence & Troup, 2018).

Neural chronometry of AB implies that distinct ERP components from EEG correspond to separate stages of information processing. Early stages of sensory processing are associated with the P1, N1, N170 (N1 component linked to processing of faces), and N2pc (N2 component linked to selective attention) components, typically seen in posterior or sensory regions. In contrast, later stages of strategic processing, such as engagement and disengagement processes, are linked to P2, N2, and P3 components, usually detected in anterior or frontal areas (Carlson, 2021; Gupta et al., 2019; Torrence & Troup, 2018).

Recent literature reviews indicate a growing interest in using ERPs as an AB outcome measure, with some components showing potential for valid and reliable measurements (Carlson, 2021; Torrence & Troup, 2018). For instance, the N2pc component has been recognized as a more dependable outcome measure than reaction time (Kappenman et al., 2015; Reutter et al., 2017). However, the relevance of certain ERP components as indices of AB remains questionable. One study found no connection between the N2pc component and trait anxiety (Kappenman et al., 2014) . The P1 component has also been investigated, yielding inconsistent results (Carlson, 2021). Additional research is needed to corroborate and generalize these findings.

fMRI has been used to investigate neural activation patterns connected to AB in populations experiencing anxiety, depression and those who are healthy (Britton et al., 2013; Hilland et al., 2020; Monk et al., 2006; Price et al., 2014). These studies have linked AB to activation in the limbic regions, anterior cingulate cortex (ACC), and prefrontal cortex (PFC). One study using an fMRI slow event dot-probe paradigm found reduced activation in the bilateral parahippocampal/hippocampal limbic region for non-anxious participants during incongruent trials, while anxious participants showed heightened activation during the same trials. A decrease in rdACC activity was observed for both groups during incongruent trials, suggesting that anxious individuals may have more difficulty regulating limbic responses when attention is shifted away from threats (Price et al., 2014).

Research involving healthy participants performing the dot-probe task has revealed consistent activation in the ventral PFC and amygdala across two separate trials. The ventral PFC was activated when participants were exposed to 500ms of face-pair stimuli, while the amygdala was activated upon exposure to 17ms of face-pair stimuli. The study was unable to differentiate between incongruent and congruent trials (Britton et al., 2013). It has been suggested that the connectivity strength between the amygdala, ACC, and PFC is positively associated with the level of AB (Carlson et al., 2014; Carlson et al., 2013).

Additional studies have shown relationships between the amygdala and visual cortex, with correlated activity when exposed to fearful faces (Morris et al., 1996; Pessoa et al., 2002). The visual cortex has also exhibited increased activity when exposed to emotional faces during the dot-probe task (Carlson et al., 2011; Pourtois et al., 2006).

In summary, investigations using EEG and fMRI techniques have so far connected the brain's emotional attention system to the amygdala, PFC, ACC and visual cortex, with the amygdala being the primary center (Torrence & Troup, 2018).

fNIRS are not able to measure the deep brain structure, but since PFC has been linked to the deeper areas, we are able to measure the ‘shadows’ of these deeper processes. The contrast maps generated from our data reveal variations in HbR concentration in different PFC regions under different conditions. These findings align with the patterns observed in fMRI studies, where AB was associated with distinct activation in limbic regions, the ACC, and the PFC (Britton et al., 2013; Hilland et al., 2020; Monk et al., 2006; Price et al., 2014). Particularly, the higher HbR concentration in the ventral medial PFC observed in our study may mirror the activation patterns of the ventral PFC seen in fMRI studies.

The fourth contrast map revealed a higher HbR concentration in the right dorsal PFC when the dot was positioned behind a face expressing emotion compared to when it was behind a neutral face. This finding aligns with studies that have suggested that the PFC plays a role in processing emotional stimuli, contributing to AB (Carlson et al., 2014; Carlson et al., 2013).

The use of fNIRS in this study offers a unique perspective in the field dominated by eye-tracking, EEG, and fMRI methods. Each of these methodologies has its strengths and limitations. For instance, ET provides valuable insights into overt behavior but may not fully capture covert attentional processes (Armstrong & Olatunji, 2012). EEG boasts high temporal resolution, but the functional significance of certain event-related potential components, such as P1 and N2pc, remains contentious (Carlson, 2021; Kappenman et al., 2014; Kappenman et al., 2015). fMRI provides detailed spatial resolution of brain activity but has limitations with regards to temporal resolution and ecological validity.

In contrast, fNIRS offers a balance between spatial and temporal resolution and is less susceptible to movement artifacts (Quaresima & Ferrari, 2019). These results might suggest that fNIRS, despite its relative novelty, could provide a valuable and cost-effective tool for investigating AB. It offers the advantage over fMRI, measuring both HbO and HbR compared to only being able to measure total Hb (Ehlis et al., 2014). Our findings underscore the promise of fNIRS in identifying neural biomarkers of AB and suggest that it may provide a valuable supplement to existing methodologies.

The current findings should be interpreted in the context of a growing body of literature advocating for a multimodal approach to measuring AB (Carlson, 2021). Such an approach could involve integrating data from ET, EEG, fMRI, and fNIRS to gain a more comprehensive understanding of the neural underpinnings of AB.

Conclusion

This study provides preliminary evidence for the detection of AB biomarkers using fNIRS. The observed hemodynamic differences in the PFC under varying conditions of the dot-probe task contribute to our understanding of the neural correlates of AB. However, further research is needed to replicate these findings and investigate their potential clinical implications in anxiety and depression.

References

Armstrong, T., & Olatunji, B. O. (2012). Eye tracking of attention in the affective disorders: A meta-analytic review and synthesis. *Clinical Psychology Review*, *32*(8), 704-723. https://doi.org/10.1016/j.cpr.2012.09.004

Barry, T. J., Vervliet, B., & Hermans, D. (2015). An integrative review of attention biases and their contribution to treatment for anxiety disorders [Review]. *Frontiers in Psychology*, *6*. https://doi.org/10.3389/fpsyg.2015.00968

Benjamini, Y., & Hochberg, Y. (1995). Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological)*, *57*(1), 289-300.

Brain-Inovation. (2023). Satori User Manual. In (1.8.2 ed.).

Britton, J. C., Bar-Haim, Y., Clementi, M. A., Sankin, L. S., Chen, G., Shechner, T., Norcross, M. A., Spiro, C. N., Lindstrom, K. M., & Pine, D. S. (2013). Training-associated changes and stability of attention bias in youth: Implications for Attention Bias Modification Treatment for pediatric anxiety. *Developmental Cognitive Neuroscience*, *4*, 52-64.

Brown, H., Eley, T., Broeren, S., Macleod, C., Rinck, M., Hadwin, J., & Lester, K. (2014). Psychometric properties of reaction time based experimental paradigms measuring anxiety-related information-processing biases in children. *Journal of anxiety disorders*, *28*(1), 97-107.

Carlson, J. M. (2021). A systematic review of event-related potentials as outcome measures of attention bias modification. *Psychophysiology*, *58*(6). https://doi.org/10.1111/psyp.13801

Carlson, J. M., Cha, J., Harmon-Jones, E., Mujica-Parodi, L. R., & Hajcak, G. (2014). Influence of the BDNF genotype on amygdalo-prefrontal white matter microstructure is linked to nonconscious attention bias to threat. *Cerebral Cortex*, *24*(9), 2249-2257.

Carlson, J. M., Cha, J., & Mujica-Parodi, L. R. (2013). Functional and structural amygdala–anterior cingulate connectivity correlates with attentional bias to masked fearful faces. *Cortex*, *49*(9), 2595-2600.

Carlson, J. M., & Fang, L. (2020). The stability and reliability of attentional bias measures in the dot-probe task: Evidence from both traditional mean bias scores and trial-level bias scores. *Motivation and Emotion*, *44*(5), 657-669. https://doi.org/10.1007/s11031-020-09834-6

Carlson, J. M., Reinke, K. S., LaMontagne, P. J., & Habib, R. (2011). Backward masked fearful faces enhance contralateral occipital cortical activity for visual targets within the spotlight of attention. *Social Cognitive and Affective Neuroscience*, *6*(5), 639-645.

Chew, P. (2015). Attentional bias: a methodological review. *Education Sciences and Psychology*, *5*, 14-29.

Delpy, D. T., Cope, M., van der Zee, P., Arridge, S., Wray, S., & Wyatt, J. (1988). Estimation of optical pathlength through tissue from direct time of flight measurement. *Physics in Medicine & Biology*, *33*(12), 1433.

Disner, S. G., Beevers, C. G., Haigh, E. A., & Beck, A. T. (2011). Neural mechanisms of the cognitive model of depression. *Nature Reviews Neuroscience*, *12*(8), 467-477.

Ehlis, A.-C., Schneider, S., Dresler, T., & Fallgatter, A. J. (2014). Application of functional near-infrared spectroscopy in psychiatry. *Neuroimage*, *85*, 478-488.

Fishburn, F. A., Ludlum, R. S., Vaidya, C. J., & Medvedev, A. V. (2019). Temporal Derivative Distribution Repair (TDDR): A motion correction method for fNIRS. *Neuroimage*, *184*, 171-179. https://doi.org/10.1016/j.neuroimage.2018.09.025

Gupta, R. S., Kujawa, A., & Vago, D. R. (2019). The neural chronometry of threat-related attentional bias: Event-related potential (ERP) evidence for early and late stages of selective attentional processing. *Int J Psychophysiol*, *146*, 20-42. https://doi.org/10.1016/j.ijpsycho.2019.08.006

Hilland, E., Landrø, N. I., Harmer, C. J., Browning, M., Maglanoc, L. A., & Jonassen, R. (2020). Attentional bias modification is associated with fMRI response toward negative stimuli in individuals with residual depression: a randomized controlled trial. *Journal of Psychiatry and Neuroscience*, *45*(1), 23-33. https://doi.org/10.1503/jpn.180118

Kappenman, E. S., Farrens, J. L., Luck, S. J., & Proudfit, G. H. (2014). Behavioral and ERP measures of attentional bias to threat in the dot-probe task: Poor reliability and lack of correlation with anxiety. *Frontiers in Psychology*, *5*, 1368.

Kappenman, E. S., MacNamara, A., & Proudfit, G. H. (2015). Electrocortical evidence for rapid allocation of attention to threat in the dot-probe task. *Social Cognitive and Affective Neuroscience*, *10*(4), 577-583.

Khan, H., Naseer, N., Yazidi, A., Eide, P. K., Hassan, H. W., & Mirtaheri, P. (2021). Analysis of Human Gait Using Hybrid EEG-fNIRS-Based BCI System: A Review [Review]. *Frontiers in Human Neuroscience*, *14*. https://doi.org/10.3389/fnhum.2020.613254

Lührs, M., Manferlotti, E., & Heinecke, A. (2022). Satori Multi-Subject GLM Guide.

Monk, C. S., Nelson, E. E., McClure, E. B., Mogg, K., Bradley, B. P., Leibenluft, E., Blair, R. J. R., Chen, G., Charney, D. S., Ernst, M., & Pine, D. S. (2006). Ventrolateral Prefrontal Cortex Activation and Attentional Bias in Response to Angry Faces in Adolescents With Generalized Anxiety Disorder. *American Journal of Psychiatry*, *163*(6), 1091-1097. https://doi.org/10.1176/ajp.2006.163.6.1091

Morris, J. S., Frith, C. D., Perrett, D. I., Rowland, D., Young, A. W., Calder, A. J., & Dolan, R. J. (1996). A differential neural response in the human amygdala to fearful and happy facial expressions. *Nature*, *383*(6603), 812-815. https://doi.org/10.1038/383812a0

Naim, R., Abend, R., Wald, I., Eldar, S., Levi, O., Fruchter, E., Ginat, K., Halpern, P., Sipos, M. L., Adler, A. B., Bliese, P. D., Quartana, P. J., Pine, D. S., & Bar-Haim, Y. (2015). Threat-Related Attention Bias Variability and Posttraumatic Stress. *American Journal of Psychiatry*, *172*(12), 1242-1250. https://doi.org/10.1176/appi.ajp.2015.14121579

Pessoa, L., Kastner, S., & Ungerleider, L. G. (2002). Attentional control of the processing of neutral and emotional stimuli. *Cognitive Brain Research*, *15*(1), 31-45.

Pinti, P., Scholkmann, F., Hamilton, A., Burgess, P., & Tachtsidis, I. (2019). Current Status and Issues Regarding Pre-processing of fNIRS Neuroimaging Data: An Investigation of Diverse Signal Filtering Methods Within a General Linear Model Framework. *Frontiers in Human Neuroscience*, *12*. https://doi.org/10.3389/fnhum.2018.00505

Pourtois, G., Schwartz, S., Seghier, M. L., Lazeyras, F., & Vuilleumier, P. (2006). Neural systems for orienting attention to the location of threat signals: an event-related fMRI study. *Neuroimage*, *31*(2), 920-933.

Price, R. B., Kuckertz, J. M., Siegle, G. J., Ladouceur, C. D., Silk, J. S., Ryan, N. D., Dahl, R. E., & Amir, N. (2015). Empirical recommendations for improving the stability of the dot-probe task in clinical research. *Psychological Assessment*, *27*(2), 365-376. https://doi.org/10.1037/pas0000036

Price, R. B., Siegle, G. J., Silk, J. S., Ladouceur, C. D., McFarland, A., Dahl, R. E., & Ryan, N. D. (2014). LOOKING UNDER THE HOOD OF THE DOT-PROBE TASK: AN fMRI STUDY IN ANXIOUS YOUTH. *Depression and Anxiety*, *31*(3), 178-187. https://doi.org/10.1002/da.22255

Quaresima, V., & Ferrari, M. (2019). Functional Near-Infrared Spectroscopy (fNIRS) for Assessing Cerebral Cortex Function During Human Behavior in Natural/Social

Situations: A Concise Review. *Organizational Research Methods*, *22*(1), 46-68. https://doi.org/10.1177/1094428116658959

Reutter, M., Hewig, J., Wieser, M. J., & Osinsky, R. (2017). The N2pc component reliably captures attentional bias in social anxiety. *Psychophysiology*, *54*(4), 519-527. https://doi.org/10.1111/psyp.12809

Schmukle, S. C. (2005). Unreliability of the dot probe task. *European Journal of Personality*, *19*(7), 595-605.

Staugaard, S. R. (2009). Reliability of two versions of the dot-probe task using photographic faces. *Psychology Science Quarterly*, *51*(3), 339-350.

Torrence, R. D. (2015). PREFRONTAL CORTEX ACTIVITY DURING ATTENTIONAL BIAS CONDITIONING WITH FEARFUL FACES: A NEAR-INFRARED SPECTROSCOPY ANALYSIS.

Torrence, R. D., & Troup, L. J. (2018). Event-related potentials of attentional bias toward faces in the dot-probe task: A systematic review. *Psychophysiology*, *55*(6), e13051. https://doi.org/10.1111/psyp.13051

WHO, V., Sergey. (2023). *Mental Health*. World Health Organisation. https://www.who.int/health-topics/mental-health#tab=tab_2

Caption List

Fig. 1 (a) fNIRS montage setup and (b) experimental setup.

Fig. 2 Illustration of the experiment paradigm.

Fig. 3 Illustration of significant differences in contras map 2: (a) 3D and (b) 2D.

Fig. 4 Illustration of significant differences in contras map 4: (a) 3D and (b) 2D

A.3 fNIRS/ET PsychoPy Code

```
# --- Import packages ---
from psychopy import locale_setup
from psychopy import prefs
from psychopy import sound, gui, visual, core, data, event, logging, clock, colors,
layout, parallel
from psychopy.constants import (NOT_STARTED, STARTED, PLAYING, PAUSED,
                                STOPPED, FINISHED, PRESSED, RELEASED, FOREVER)

import numpy as np  # whole numpy lib is available, prepend 'np.'
from numpy import (sin, cos, tan, log, log10, pi, average,
                   sqrt, std, deg2rad, rad2deg, linspace, asarray)
from numpy.random import random, randint, normal, shuffle, choice as randchoice
import os  # handy system and path functions
import sys  # to get file system encoding

import psychopy.iohub as io
from psychopy.hardware import keyboard

# Run 'Before Experiment' code from Start_and_end_code
import __future__
import pylink
import os
import platform
import random
import time
import sys
from EyeLinkCoreGraphicsPsychoPy import EyeLinkCoreGraphicsPsychoPy
from psychopy import visual, core, event, monitors, gui
from PIL import Image  # for preparing the Host backdrop image
from string import ascii_letters, digits



#Setting up gloabl variable for reaction time and vas-scale
global Reaction_Time_Display
Reaction_Time_Display = 0
global Reaction_Time_Record
Reaction_Time_Record = 10
global VAS
global trigger_sent
trigger_sent = False

# Switch to the script folder
script_path = os.path.dirname(sys.argv[0])
if len(script_path) != 0:
    os.chdir(script_path)

# Show only critical log message in the PsychoPy console
from psychopy import logging
logging.console.setLevel(logging.CRITICAL)

#List to later be stored in excel document (this is the heading)
```

```
excel_list =
[['Trial_Number','Participant_ID','Face_Stimuli_Left','Face_Stimuli_Right','Type','Face
_Pairs','Gender','Dot_Location','Response_Accuracy','Reaction_Time','Experiemnt_Ti
me', 'Time_Jitter', 'VAS']]



#Variable  that is later combined with timer to decide when and what dot will show
show_dot_left = False
show_dot_right = False



Trial_Number = 0



# Set this variable to True if you use the built-in retina screen as your
# primary display device on macOS. If have an external monitor, set this
# variable True if you choose to "Optimize for Built-in Retina Display"
# in the Displays preference settings.
use_retina = False

# Set this variable to True to run the script in "Dummy Mode"
dummy_mode = False

# Set this variable to True to run the task in full screen mode
# It is easier to debug the script in non-fullscreen mode
full_screen = True
```

```
# Set up EDF data file name and local data folder
#
# The EDF data filename should not exceed 8 alphanumeric characters
# use ONLY number 0-9, letters, & _ (underscore) in the filename
edf_fname = ''

# Prompt user to specify an EDF data filename
# before we open a fullscreen window
dlg_title = 'Enter ParticipantID'
dlg_prompt = 'Please enter a file name with 8 or fewer characters\n' + \
            '[letters, numbers, and underscore].'

# loop until we get a valid filename
while True:
    dlg = gui.Dlg(dlg_title)
    dlg.addText(dlg_prompt)
    dlg.addField('ParticipantID:', edf_fname)
    # show dialog and wait for OK or Cancel
    ok_data = dlg.show()
    if dlg.OK:  # if ok_data is not None
        print('EDF data filename: {}'.format(ok_data[0]))
    else:
        print('user cancelled')
        core.quit()
        sys.exit()
    # get the string entered by the experimenter
    tmp_str = dlg.data[0]
    # strip trailing characters, ignore the ".edf" extension
    edf_fname = tmp_str.rstrip().split('.')[0]

    # check if the filename is valid (length <= 8 & no special char)
    allowed_char = ascii_letters + digits + '_'
    if not all([c in allowed_char for c in edf_fname]):
        print('ERROR: Invalid EDF filename')
    elif len(edf_fname) > 8:
        print('ERROR: EDF filename should not exceed 8 characters')
    else:
        break


# Set up a folder to store the EDF data files and the associated resources
# e.g., files defining the interest areas used in each trial
results_folder = 'Eye_FNIRS'
if not os.path.exists(results_folder):
    os.makedirs(results_folder)

# We download EDF data file from the EyeLink Host PC to the local hard
# drive at the end of each testing session, here we rename the EDF to
# include session start date/time
time_str = time.strftime("_%Y_%m_%d_%H_%M", time.localtime())
```

```
session_identifier = edf_fname + "_FNIRS" + time_str


# create a folder for the current testing session in the "results" folder
session_folder = os.path.join(results_folder, session_identifier)
if not os.path.exists(session_folder):
    os.makedirs(session_folder)


# Step 1: Connect to the EyeLink Host PC
#
# The Host IP address, by default, is "100.1.1.1".
# the "el_tracker" objected created here can be accessed through the Pylink
# Set the Host PC address to "None" (without quotes) to run the script
# in "Dummy Mode"
if dummy_mode:
    el_tracker = pylink.EyeLink(None)
else:
    try:
        el_tracker = pylink.EyeLink("100.1.1.1")
    except RuntimeError as error:
        print('ERROR:', error)
        core.quit()
        sys.exit()


# Step 2: Open an EDF data file on the Host PC
edf_file = edf_fname + "_FNIRS" + ".EDF"
```

```
try:
    el_tracker.openDataFile(edf_file)
except RuntimeError as err:
    print('ERROR:', err)
    # close the link if we have one open
    if el_tracker.isConnected():
        el_tracker.close()
    core.quit()
    sys.exit()

# Add a header text to the EDF file to identify the current experiment name
# This is OPTIONAL. If your text starts with "RECORDED BY " it will be
# available in DataViewer's Inspector window by clicking
# the EDF session node in the top panel and looking for the "Recorded By:"
# field in the bottom panel of the Inspector.
preamble_text = 'RECORDED BY %s' % os.path.basename(__file__)
el_tracker.sendCommand("add_file_preamble_text '%s'" % preamble_text)

# Step 3: Configure the tracker
#
# Put the tracker in offline mode before we change tracking parameters
el_tracker.setOfflineMode()

# Get the software version:  1-EyeLink I, 2-EyeLink II, 3/4-EyeLink 1000,
# 5-EyeLink 1000 Plus, 6-Portable DUO
```

```
eyelink_ver = 0  # set version to 0, in case running in Dummy mode
if not dummy_mode:
    vstr = el_tracker.getTrackerVersionString()
    eyelink_ver = int(vstr.split()[-1].split('.')[0])
    # print out some version info in the shell
    print('Running experiment on %s, version %d' % (vstr, eyelink_ver))

# File and Link data control
# what eye events to save in the EDF file, include everything by default
file_event_flags =
'LEFT,RIGHT,FIXATION,SACCADE,BLINK,MESSAGE,BUTTON,INPUT'
# what eye events to make available over the link, include everything by default
link_event_flags =
'LEFT,RIGHT,FIXATION,SACCADE,BLINK,BUTTON,FIXUPDATE,INPUT'
# what sample data to save in the EDF data file and to make available
# over the link, include the 'HTARGET' flag to save head target sticker
# data for supported eye trackers
if eyelink_ver > 3:
    file_sample_flags =
'LEFT,RIGHT,GAZE,HREF,RAW,AREA,HTARGET,GAZERES,BUTTON,STATUS,IN
PUT'
    link_sample_flags =
'LEFT,RIGHT,GAZE,GAZERES,AREA,HTARGET,STATUS,INPUT'
else:
    file_sample_flags =
'LEFT,RIGHT,GAZE,HREF,RAW,AREA,GAZERES,BUTTON,STATUS,INPUT'
    link_sample_flags = 'LEFT,RIGHT,GAZE,GAZERES,AREA,STATUS,INPUT'
```

```
el_tracker.sendCommand("file_event_filter = %s" % file_event_flags)
el_tracker.sendCommand("file_sample_data = %s" % file_sample_flags)
el_tracker.sendCommand("link_event_filter = %s" % link_event_flags)
el_tracker.sendCommand("link_sample_data = %s" % link_sample_flags)

# Optional tracking parameters
# Sample rate, 250, 500, 1000, or 2000, check your tracker specification
# if eyelink_ver > 2:
#     el_tracker.sendCommand("sample_rate 1000")
# Choose a calibration type, H3, HV3, HV5, HV13 (HV = horizontal/vertical),
el_tracker.sendCommand("calibration_type = HV9")
# Set a gamepad button to accept calibration/drift check target
# You need a supported gamepad/button box that is connected to the Host PC
el_tracker.sendCommand("button_function 5 'accept_target_fixation'")

# Step 4: set up a graphics environment for calibration
#
# Open a window, be sure to specify monitor parameters
#mon = monitors.Monitor('myMonitor', width=53.0, distance=115.0)
win = visual.Window(size=(1920,1080),
                    pos=(1920,0),
                    winType='pyglet',
                    units='pix')

# get the native screen resolution used by PsychoPy
scn_width, scn_height = win.size
```

```
# resolution fix for Mac retina displays
if 'Darwin' in platform.system():
    if use_retina:
        scn_width = int(scn_width/2.0)
        scn_height = int(scn_height/2.0)

# Pass the display pixel coordinates (left, top, right, bottom) to the tracker
# see the EyeLink Installation Guide, "Customizing Screen Settings"
el_coords = "screen_pixel_coords = 0 0 %d %d" % (scn_width - 1, scn_height - 1)
el_tracker.sendCommand(el_coords)

# Write a DISPLAY_COORDS message to the EDF file
# Data Viewer needs this piece of info for proper visualization, see Data
# Viewer User Manual, "Protocol for EyeLink Data to Viewer Integration"
dv_coords = "DISPLAY_COORDS  0 0 %d %d" % (scn_width - 1, scn_height - 1)
el_tracker.sendMessage(dv_coords)

# Configure a graphics environment (genv) for tracker calibration
genv = EyeLinkCoreGraphicsPsychoPy(el_tracker, win)
print(genv)  # print out the version number of the CoreGraphics library

# Set background and foreground colors for the calibration target
# in PsychoPy, (-1, -1, -1)=black, (1, 1, 1)=white, (0, 0, 0)=mid-gray
foreground_color = (-1, -1, -1)
background_color = win.color
genv.setCalibrationColors(foreground_color, background_color)
```

```
# Set up the calibration target
#
# The target could be a "circle" (default), a "picture", a "movie" clip,
# or a rotating "spiral". To configure the type of calibration target, set
# genv.setTargetType to "circle", "picture", "movie", or "spiral", e.g.,
# genv.setTargetType('picture')
#
# Use gen.setPictureTarget() to set a "picture" target
# genv.setPictureTarget(os.path.join('images', 'fixTarget.bmp'))
#
# Use genv.setMovieTarget() to set a "movie" target
# genv.setMovieTarget(os.path.join('videos', 'calibVid.mov'))

# Use a picture as the calibration target
genv.setTargetType('picture')
genv.setPictureTarget(os.path.join('images', 'fixTarget.bmp'))

# Configure the size of the calibration target (in pixels)
# this option applies only to "circle" and "spiral" targets
# genv.setTargetSize(24)

# Beeps to play during calibration, validation and drift correction
# parameters: target, good, error
#     target -- sound to play when target moves
#     good -- sound to play on successful operation
#     error -- sound to play on failure or interruption
# Each parameter could be ''--default sound, 'off'--no sound, or a wav file
genv.setCalibrationSounds('', '', '')

# resolution fix for macOS retina display issues
if use_retina:
    genv.fixMacRetinaDisplay()

# Request Pylink to use the PsychoPy window we opened above for calibration
pylink.openGraphicsEx(genv)


# define a few helper functions for trial handling


def clear_screen(win):
    """ clear up the PsychoPy window"""

    win.fillColor = genv.getBackgroundColor()
    win.flip()


def show_msg(win, text, wait_for_keypress=True):
    """ Show task instructions on screen"""

    msg = visual.TextStim(win, text,
                     color=genv.getForegroundColor(),
                     wrapWidth=scn_width/2)
    clear_screen(win)
    msg.draw()
    win.flip()

    # wait indefinitely, terminates upon any key press
    if wait_for_keypress:
        event.waitKeys()
        clear_screen(win)


def terminate_task():
    """ Terminate the task gracefully and retrieve the EDF data file

    file_to_retrieve: The EDF on the Host that we would like to download
    win: the current window used by the experimental script
    """

    el_tracker = pylink.getEYELINK()

    if el_tracker.isConnected():
        # Terminate the current trial first if the task terminated prematurely
        error = el_tracker.isRecording()
        if error == pylink.TRIAL_OK:
            abort_trial()
```

```
# Put tracker in Offline mode
el_tracker.setOfflineMode()

# Clear the Host PC screen and wait for 500 ms
el_tracker.sendCommand('clear_screen 0')
pylink.msecDelay(500)

# Close the edf data file on the Host
el_tracker.closeDataFile()

# Show a file transfer message on the screen
msg = 'EDF data is transferring from EyeLink Host PC...'
show_msg(win, msg, wait_for_keypress=False)

# Download the EDF data file from the Host PC to a local data folder
# parameters: source_file_on_the_host, destination_file_on_local_drive
local_edf = os.path.join(session_folder, session_identifier + '.EDF')
try:
    el_tracker.receiveDataFile(edf_file, local_edf)
except RuntimeError as error:
    print('ERROR:', error)

# Close the link to the tracker.
el_tracker.close()
```

```
    # close the PsychoPy window
    win.close()

    # quit PsychoPy
    core.quit()
    sys.exit()


def abort_trial():
    """Ends recording """

    el_tracker = pylink.getEYELINK()

    # Stop recording
    if el_tracker.isRecording():
        # add 100 ms to catch final trial events
        pylink.pumpDelay(100)
        el_tracker.stopRecording()

    # clear the screen
    clear_screen(win)
    # Send a message to clear the Data Viewer screen
    bgcolor_RGB = (116, 116, 116)
    el_tracker.sendMessage('!V CLEAR %d %d %d' % bgcolor_RGB)

    # send a message to mark trial end
    el_tracker.sendMessage('TRIAL_RESULT %d' % pylink.TRIAL_ERROR)

    return pylink.TRIAL_ERROR

visual.Window(size=(1920,1080),pos=(1920,0),screen=2)

# Step 5: Set up the camera and calibrate the tracker

# Show the task instructions
task_msg = 'Ready for eye tracker calibration\n'
if dummy_mode:
    task_msg = task_msg + '\nNow, press ENTER to start the task'
else:
    task_msg = task_msg + '\nNow, press ENTER twice to calibrate tracker'
show_msg(win, task_msg)

# skip this step if running the script in Dummy Mode
if not dummy_mode:
    try:
        el_tracker.doTrackerSetup()
    except RuntimeError as err:
        print('ERROR:', err)
        el_tracker.exitCalibration()
```

```
# Ensure that relative paths start from the same directory as this script
_thisDir = os.path.dirname(os.path.abspath(__file__))
os.chdir(_thisDir)
# Store info about the experiment session
psychopyVersion = '2022.2.4'
expName = 'FNIRS_DP'  # from the Builder filename that created this script
expInfo = {}
# --- Show participant info dialog --
dlg = gui.DlgFromDict(dictionary=expInfo, sortKeys=False, title=expName)
if dlg.OK == False:
    core.quit()  # user pressed cancel
expInfo['date'] = data.getDateStr()  # add a simple timestamp
expInfo['expName'] = expName
expInfo['psychopyVersion'] = psychopyVersion

# Data file name stem = absolute path + name; later add .psyexp, .csv, .log, etc
filename = _thisDir + os.sep + u'data/%s_%s_%s' % (edf_fname, expName,
expInfo['date'])

# An ExperimentHandler isn't essential but helps with data saving
thisExp = data.ExperimentHandler(name=expName, version='',
    extraInfo=expInfo, runtimeInfo=None,
    originPath='C:\\Users\\Sven Ivar Ougendal\\OneDrive - OsloMet\\Master - Brain
Health projects - Sandra Klonteig et al. - Sven Master Thesis\\3 - Experiment
paradigm\\FNIRS_Data_Psychopy\\FNIRS_Psychopy.py',
    savePickle=True, saveWideText=False,
    dataFileName=filename)
```

```
# save a log file for detail verbose info
logFile = logging.LogFile(filename+'.log', level=logging.EXP)
logging.console.setLevel(logging.WARNING)  # this outputs to the screen, not a file

endExpNow = False  # flag for 'escape' or other condition => quit the exp
frameTolerance = 0.001  # how close to onset before 'same' frame

# Start Code - component code to be run after the window creation

# --- Setup the Window ---
win = visual.Window(
    size=[1920, 1080], fullscr=True, screen=1,
    winType='pyglet', allowStencil=False,
    monitor='testMonitor', color='black', colorSpace='rgb',
    blendMode='avg', useFBO=True,
    units='height')
win.mouseVisible = False
# store frame rate of monitor if we can measure it
expInfo['frameRate'] = win.getActualFrameRate()
if expInfo['frameRate'] != None:
    frameDur = 1.0 / round(expInfo['frameRate'])
else:
    frameDur = 1.0 / 60.0  # could not measure, so guess
# --- Setup input devices ---
ioConfig = {}
```

```
# Setup iohub keyboard
ioConfig['Keyboard'] = dict(use_keymap='psychopy')

ioSession = '1'
if 'session' in expInfo:
    ioSession = str(expInfo['session'])
ioServer = io.launchHubServer(window=win, **ioConfig)
eyetracker = None

# create a default keyboard (e.g. to check for escape)
defaultKeyboard = keyboard.Keyboard(backend='iohub')

# --- Initialize components for Routine "start_and_eyetracking_calibration" ---
# Run 'Begin Experiment' code from Start_and_end_code
#Fetching ID for the excel document
Participant_ID = edf_fname

#Functin to map cordianates from psychopy
def map_cord(x,y):
    return (x-scn_width/2, y-scn_height/2)



#Setting up LSL for sending triggers (for fNIRS)
from pylsl import StreamInfo, StreamOutlet # import required classes
```

```
info = StreamInfo( 'TriggerStream', type='Markers', channel_count=1,
channel_format='int32', source_id='Example') # sets variables for object info
outlet = StreamOutlet(info)



#Code for blockdesign
global random_blocks_counter
random_blocks_counter = 0


global list_random_blocks
list_random_blocks = []
a = 0
b = 4


for i in range(20):
    list_random_blocks.append(range(a, b))
    a = b
    b = a + 4


random.shuffle(list_random_blocks)




#list_random_blocks = [(0,3), (4, 7), (8, 11), (12, 15), (16, 19), (20, 23), (24, 27), (28,
31), (32, 35), (36, 39), (40, 43), (44, 47), (48, 51), (52, 55), (56, 59), (60, 63), (64, 67),
(68, 71), (72, 75), (76, 79)]
```

```
# --- Initialize components for Routine "vas_measure" ---
vas_text = visual.TextStim(win=win, name='vas_text',
    text='From a scale from 0 - 9, how tired are you?\n(Higher number = more tired)',
    font='Open Sans',
    pos=(0, 0), height=0.05, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=-1.0);
vas_response = keyboard.Keyboard()

# --- Initialize components for Routine "start_fnirs_reminder" ---
reminder_eeg_data_collection_text = visual.TextStim(win=win,
name='reminder_eeg_data_collection_text',
    text='Researcher will start data collection now\n\nThe experiment will then start
soon',
    font='Open Sans',
    pos=(0, 0), height=0.05, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=0.0);
eeg_data_collection_on_key = keyboard.Keyboard()

# --- Initialize components for Routine "dot_probe_task_info" ---
task_info_text = visual.TextStim(win=win, name='task_info_text',
    text="The next task goes as follows:\n1. A fixation cross will apear, followed by two
pictures on each side of the screen. \n2. After a short time, the pictures will disapear
and a DOT will apear on either side of the screen. \n3. You objective is to press 'L' or
'R' as fast a possible when the DOT apears. \n4. Your reaction time will apear on the
screen.\n5. Then the task repeats itself. It will take aproximatly 10min\n6. You will
then get informaition on the next task\n\nTo get ready, but your left indexfinger on 'L'
and right indexfinger on 'R'\nPress 'L' or 'R' to start the task",
    font='Open Sans',
    pos=(0, 0), height=0.04, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=0.0);
task_info_response = keyboard.Keyboard()

# --- Initialize components for Routine "dot_probe_start_counter" ---
dot_probe_text_counter = visual.TextStim(win=win, name='dot_probe_text_counter',
    text='',
    font='Open Sans',
    pos=(0, -0.1), height=0.1, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=0.0);
dot_probe_start_text = visual.TextStim(win=win, name='dot_probe_start_text',
    text='Relax. The test starts in:',
    font='Open Sans',
    pos=(0, 0.1), height=0.1, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=-1.0);
```

```
# --- Initialize components for Routine "Fixation_Cross" ---
fixation_cross = visual.ShapeStim(
    win=win, name='fixation_cross', vertices='cross',units='pix',
    size=(35, 35),
    ori=0.0, pos=(0, 0), anchor='center',
    lineWidth=0.02,     colorSpace='rgb',  lineColor='white', fillColor='white',
    opacity=None, depth=-1.0, interpolate=True)

# --- Initialize components for Routine "Faces_Stimuli" ---
image_left = visual.ImageStim(
    win=win,
    name='image_left', units='pix',
    image='sin', mask=None, anchor='center',
    ori=0.0, pos=(-687, 0), size=(362, 506),
    color=[1,1,1], colorSpace='rgb', opacity=None,
    flipHoriz=False, flipVert=False,
    texRes=128.0, interpolate=True, depth=-1.0)
image_right = visual.ImageStim(
    win=win,
    name='image_right', units='pix',
    image='sin', mask=None, anchor='center',
    ori=0.0, pos=(687, 0), size=(362, 506),
    color=[1,1,1], colorSpace='rgb', opacity=None,
    flipHoriz=False, flipVert=False,
    texRes=128.0, interpolate=True, depth=-2.0)
fixation_cross_2 = visual.ShapeStim(
    win=win, name='fixation_cross_2', vertices='cross',units='pix',
    size=(35, 35),
    ori=0.0, pos=(0, 0), anchor='center',
    lineWidth=0.02,     colorSpace='rgb',  lineColor='white', fillColor='white',
    opacity=None, depth=-3.0, interpolate=True)

# --- Initialize components for Routine "Dot_Stimuli" ---
key_reaction = keyboard.Keyboard()
dot1_left = visual.ShapeStim(
    win=win, name='dot1_left',units='pix',
    size=(25, 25), vertices='circle',
    ori=0.0, pos=(-687, 0), anchor='center',
    lineWidth=1.0,     colorSpace='rgb',  lineColor='white', fillColor='white',
    opacity=None, depth=-2.0, interpolate=True)
dot1_right = visual.ShapeStim(
    win=win, name='dot1_right',units='pix',
    size=(25, 25), vertices='circle',
    ori=0.0, pos=(687, 0), anchor='center',
    lineWidth=1.0,     colorSpace='rgb',  lineColor='white', fillColor='white',
    opacity=None, depth=-3.0, interpolate=True)
fixation_cross_3 = visual.ShapeStim(
    win=win, name='fixation_cross_3', vertices='cross',units='pix',
    size=(35, 35),
    ori=0.0, pos=(0, 0), anchor='center',
    lineWidth=0.02,     colorSpace='rgb',  lineColor='white', fillColor='white',
    opacity=None, depth=-4.0, interpolate=True)
```

```
# --- Initialize components for Routine "Rest" ---
reaction_time_text = visual.TextStim(win=win, name='reaction_time_text',
    text=None,
    font='Open Sans',
    pos=(0, 0), height=0.05, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=-1.0);

# --- Initialize components for Routine "inter_trial_interval" ---
text = visual.TextStim(win=win, name='text',
    text=None,
    font='Open Sans',
    pos=(0, 0), height=0.05, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=-1.0);
reaction_time_text_2 = visual.TextStim(win=win, name='reaction_time_text_2',
    text='Your best reaction time:',
    font='Open Sans',
    pos=(0, 0), height=0.05, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=-2.0);
```

```
reaction_time_numbers_2 = visual.TextStim(win=win,
name='reaction_time_numbers_2',
    text='',
    font='Open Sans',
    pos=(0, -0.1), height=0.05, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=-3.0);

# --- Initialize components for Routine "experiment_ended" ---
dot_probe_ended_text = visual.TextStim(win=win, name='dot_probe_ended_text',
    text='Experiment is over\n\nThank you ',
    font='Open Sans',
    pos=(0, 0), height=0.05, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=0.0);

# Create some handy timers
globalClock = core.Clock()  # to track the time since experiment started
routineTimer = core.Clock()  # to track time remaining of each (possibly non-slip)
routine

# --- Prepare to start Routine "start_and_eyetracking_calibration" ---
continueRoutine = True
routineForceEnded = False
```

```
# update component parameters for each repeat
# keep track of which components have finished
start_and_eyetracking_calibrationComponents = []
for thisComponent in start_and_eyetracking_calibrationComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "start_and_eyetracking_calibration" ---
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in start_and_eyetracking_calibrationComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished

    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()

# --- Ending Routine "start_and_eyetracking_calibration" ---
for thisComponent in start_and_eyetracking_calibrationComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
# Run 'End Routine' code from Start_and_end_code
#Hides the mouse when the experiment is running
event.Mouse(visible=False)
```

```
# the Routine "start_and_eyetracking_calibration" was not non-slip safe, so reset the
non-slip timer
routineTimer.reset()


# --- Prepare to start Routine "vas_measure" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
vas_response.keys = []
vas_response.rt = []
_vas_response_allKeys = []
# keep track of which components have finished
vas_measureComponents = [vas_text, vas_response]
for thisComponent in vas_measureComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1


# --- Run Routine "vas_measure" ---
```

```
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # *vas_text* updates
    if vas_text.status == NOT_STARTED and tThisFlip >= 0.0-frameTolerance:
        # keep track of start time/frame for later
        vas_text.frameNStart = frameN  # exact frame index
        vas_text.tStart = t  # local t and not account for scr refresh
        vas_text.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(vas_text, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'vas_text.started')
        vas_text.setAutoDraw(True)

    # *vas_response* updates
    waitOnFlip = False
    if vas_response.status == NOT_STARTED and tThisFlip >= 0.0-frameTolerance:
        # keep track of start time/frame for later
        vas_response.frameNStart = frameN  # exact frame index
        vas_response.tStart = t  # local t and not account for scr refresh
        vas_response.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(vas_response, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'vas_response.started')
        vas_response.status = STARTED
        # keyboard checking is just starting
        waitOnFlip = True
        win.callOnFlip(vas_response.clock.reset)  # t=0 on next screen flip
        win.callOnFlip(vas_response.clearEvents, eventType='keyboard')  # clear
events on next screen flip
    if vas_response.status == STARTED and not waitOnFlip:
        theseKeys = vas_response.getKeys(keyList=['1','2','3','4','5','6','7','8','9'],
waitRelease=False)
        _vas_response_allKeys.extend(theseKeys)
        if len(_vas_response_allKeys):
            vas_response.keys = _vas_response_allKeys[-1].name  # just the last key
pressed
            vas_response.rt = _vas_response_allKeys[-1].rt
            # a response ends the routine
            continueRoutine = False

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
```

```
            break
        continueRoutine = False  # will revert to True if at least one component still running
        for thisComponent in vas_measureComponents:
            if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
                continueRoutine = True
                break  # at least one component has not yet finished

        # refresh the screen
        if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
            win.flip()

    # --- Ending Routine "vas_measure" ---
    for thisComponent in vas_measureComponents:
        if hasattr(thisComponent, "setAutoDraw"):
            thisComponent.setAutoDraw(False)
    # Run 'End Routine' code from vas_code
    VAS = vas_response.keys
    # check responses
    if vas_response.keys in ['', [], None]:  # No response was made
        vas_response.keys = None
    thisExp.addData('vas_response.keys',vas_response.keys)
    if vas_response.keys != None:  # we had a response
        thisExp.addData('vas_response.rt', vas_response.rt)
    thisExp.nextEntry()
    # the Routine "vas_measure" was not non-slip safe, so reset the non-slip timer
    routineTimer.reset()
```

```
# --- Prepare to start Routine "start_fnirs_reminder" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
eeg_data_collection_on_key.keys = []
eeg_data_collection_on_key.rt = []
_eeg_data_collection_on_key_allKeys = []
# keep track of which components have finished
start_fnirs_reminderComponents = [reminder_eeg_data_collection_text,
eeg_data_collection_on_key]
for thisComponent in start_fnirs_reminderComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "start_fnirs_reminder" ---
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # *reminder_eeg_data_collection_text* updates
    if reminder_eeg_data_collection_text.status == NOT_STARTED and tThisFlip >=
0.0-frameTolerance:
        # keep track of start time/frame for later
        reminder_eeg_data_collection_text.frameNStart = frameN  # exact frame index
        reminder_eeg_data_collection_text.tStart = t  # local t and not account for scr
refresh
        reminder_eeg_data_collection_text.tStartRefresh = tThisFlipGlobal  # on global
time
        win.timeOnFlip(reminder_eeg_data_collection_text, 'tStartRefresh')  # time at
next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'reminder_eeg_data_collection_text.started')
        reminder_eeg_data_collection_text.setAutoDraw(True)

    # *eeg_data_collection_on_key* updates
    waitOnFlip = False
    if eeg_data_collection_on_key.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
        # keep track of start time/frame for later
        eeg_data_collection_on_key.frameNStart = frameN  # exact frame index
        eeg_data_collection_on_key.tStart = t  # local t and not account for scr refresh
        eeg_data_collection_on_key.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(eeg_data_collection_on_key, 'tStartRefresh')  # time at next scr
refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'eeg_data_collection_on_key.started')
        eeg_data_collection_on_key.status = STARTED
        # keyboard checking is just starting
        waitOnFlip = True
        win.callOnFlip(eeg_data_collection_on_key.clock.reset)  # t=0 on next screen
flip
        win.callOnFlip(eeg_data_collection_on_key.clearEvents, eventType='keyboard')
# clear events on next screen flip
    if eeg_data_collection_on_key.status == STARTED and not waitOnFlip:
        theseKeys = eeg_data_collection_on_key.getKeys(keyList=['space'],
waitRelease=False)
        _eeg_data_collection_on_key_allKeys.extend(theseKeys)
        if len(_eeg_data_collection_on_key_allKeys):
            eeg_data_collection_on_key.keys = _eeg_data_collection_on_key_allKeys[-
1].name  # just the last key pressed
            eeg_data_collection_on_key.rt = _eeg_data_collection_on_key_allKeys[-1].rt
            # a response ends the routine
            continueRoutine = False

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()
    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in start_fnirs_reminderComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished

    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()

# --- Ending Routine "start_fnirs_reminder" ---
for thisComponent in start_fnirs_reminderComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
# check responses
if eeg_data_collection_on_key.keys in ['', [], None]:  # No response was made
    eeg_data_collection_on_key.keys = None
thisExp.addData('eeg_data_collection_on_key.keys',eeg_data_collection_on_key.ke
ys)
if eeg_data_collection_on_key.keys != None:  # we had a response
    thisExp.addData('eeg_data_collection_on_key.rt', eeg_data_collection_on_key.rt)
thisExp.nextEntry()
# the Routine "start_fnirs_reminder" was not non-slip safe, so reset the non-slip timer
routineTimer.reset()

# --- Prepare to start Routine "dot_probe_task_info" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
task_info_response.keys = []
task_info_response.rt = []
_task_info_response_allKeys = []
# keep track of which components have finished
dot_probe_task_infoComponents = [task_info_text, task_info_response]
for thisComponent in dot_probe_task_infoComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "dot_probe_task_info" ---
```

```
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # *task_info_text* updates
    if task_info_text.status == NOT_STARTED and tThisFlip >= 0.0-frameTolerance:
        # keep track of start time/frame for later
        task_info_text.frameNStart = frameN  # exact frame index
        task_info_text.tStart = t  # local t and not account for scr refresh
        task_info_text.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(task_info_text, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'task_info_text.started')
        task_info_text.setAutoDraw(True)

    # *task_info_response* updates
    waitOnFlip = False
    if task_info_response.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
        # keep track of start time/frame for later
        task_info_response.frameNStart = frameN  # exact frame index
        task_info_response.tStart = t  # local t and not account for scr refresh
        task_info_response.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(task_info_response, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'task_info_response.started')
        task_info_response.status = STARTED
        # keyboard checking is just starting
        waitOnFlip = True
        win.callOnFlip(task_info_response.clock.reset)  # t=0 on next screen flip
        win.callOnFlip(task_info_response.clearEvents, eventType='keyboard')  # clear
events on next screen flip
    if task_info_response.status == STARTED and not waitOnFlip:
        theseKeys = task_info_response.getKeys(keyList=['1','2'], waitRelease=False)
        _task_info_response_allKeys.extend(theseKeys)
        if len(_task_info_response_allKeys):
            task_info_response.keys = _task_info_response_allKeys[-1].name  # just the
last key pressed
            task_info_response.rt = _task_info_response_allKeys[-1].rt
            # a response ends the routine
            continueRoutine = False

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
```

```
            routineForceEnded = True
            break
        continueRoutine = False  # will revert to True if at least one component still running
        for thisComponent in dot_probe_task_infoComponents:
            if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
                continueRoutine = True
                break  # at least one component has not yet finished

        # refresh the screen
        if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
            win.flip()

    # --- Ending Routine "dot_probe_task_info" ---
    for thisComponent in dot_probe_task_infoComponents:
        if hasattr(thisComponent, "setAutoDraw"):
            thisComponent.setAutoDraw(False)
    # check responses
    if task_info_response.keys in ['', [], None]:  # No response was made
        task_info_response.keys = None
    thisExp.addData('task_info_response.keys',task_info_response.keys)
    if task_info_response.keys != None:  # we had a response
        thisExp.addData('task_info_response.rt', task_info_response.rt)
    thisExp.nextEntry()
    # the Routine "dot_probe_task_info" was not non-slip safe, so reset the non-slip timer
    routineTimer.reset()
```

```
# --- Prepare to start Routine "dot_probe_start_counter" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
# Run 'Begin Routine' code from start_eye_track_recording_and_clock_code
#Trigger for start baseline for fnirs
outlet.push_sample([1])

# get a reference to the currently active EyeLink connection
el_tracker = pylink.getEYELINK()

# put the tracker in the offline mode first
el_tracker.setOfflineMode()

# clear the host screen before we draw the backdrop
el_tracker.sendCommand('clear_screen 0')

trial_index = 0



# put tracker in idle/offline mode before recording
el_tracker.setOfflineMode()

# Start recording
# arguments: sample_to_file, events_to_file, sample_over_link,
# event_over_link (1-yes, 0-no)
```

```
try:
    el_tracker.startRecording(1, 1, 1, 1)
except RuntimeError as error:
    print("ERROR:", error)
    abort_trial()



# keep track of which components have finished
dot_probe_start_counterComponents = [dot_probe_text_counter,
dot_probe_start_text]
for thisComponent in dot_probe_start_counterComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1


# --- Run Routine "dot_probe_start_counter" ---
while continueRoutine and routineTimer.getTime() < 30.0:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # *dot_probe_text_counter* updates
    if dot_probe_text_counter.status == NOT_STARTED and tThisFlip >= 0-
frameTolerance:
        # keep track of start time/frame for later
        dot_probe_text_counter.frameNStart = frameN  # exact frame index
        dot_probe_text_counter.tStart = t  # local t and not account for scr refresh
        dot_probe_text_counter.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(dot_probe_text_counter, 'tStartRefresh')  # time at next scr
refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'dot_probe_text_counter.started')
        dot_probe_text_counter.setAutoDraw(True)
    if dot_probe_text_counter.status == STARTED:
        # is it time to stop? (based on global clock, using actual start)
        if tThisFlipGlobal > dot_probe_text_counter.tStartRefresh + 30-frameTolerance:
            # keep track of stop time/frame for later
            dot_probe_text_counter.tStop = t  # not accounting for scr refresh
            dot_probe_text_counter.frameNStop = frameN  # exact frame index
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'dot_probe_text_counter.stopped')
            dot_probe_text_counter.setAutoDraw(False)
    if dot_probe_text_counter.status == STARTED:  # only update if drawing
        dot_probe_text_counter.setText(round(30.0 - t, ndigits = 0)



, log=False)


    # *dot_probe_start_text* updates
    if dot_probe_start_text.status == NOT_STARTED and tThisFlip >= 0-
frameTolerance:
        # keep track of start time/frame for later
        dot_probe_start_text.frameNStart = frameN  # exact frame index
        dot_probe_start_text.tStart = t  # local t and not account for scr refresh
        dot_probe_start_text.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(dot_probe_start_text, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'dot_probe_start_text.started')
        dot_probe_start_text.setAutoDraw(True)
    if dot_probe_start_text.status == STARTED:
        # is it time to stop? (based on global clock, using actual start)
        if tThisFlipGlobal > dot_probe_start_text.tStartRefresh + 30-frameTolerance:
            # keep track of stop time/frame for later
            dot_probe_start_text.tStop = t  # not accounting for scr refresh
            dot_probe_start_text.frameNStop = frameN  # exact frame index
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'dot_probe_start_text.stopped')
            dot_probe_start_text.setAutoDraw(False)
    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in dot_probe_start_counterComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished

    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()

# --- Ending Routine "dot_probe_start_counter" ---
for thisComponent in dot_probe_start_counterComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
# Run 'End Routine' code from start_eye_track_recording_and_clock_code
#Starting timer. It starts when the image routines begins.
expClock = core.Clock()
```

```
#send message to Dataviewer the routine is starting
el_tracker.sendMessage('beginExperiment')



#Trigger for start experiment
outlet.push_sample([1])
# using non-slip timing so subtract the expected duration of this Routine (unless ended on request)
if routineForceEnded:
    routineTimer.reset()
else:
    routineTimer.addTime(-30.000000)

# set up handler to look after randomisation of conditions etc
trials = data.TrialHandler(nReps=20.0, method='random',
    extraInfo=expInfo, originPath=-1,
    trialList=[None],
    seed=None, name='trials')
thisExp.addLoop(trials)  # add the loop to the experiment
thisTrial_2 = trials.trialList[0]  # so we can initialise stimuli with some values
# abbreviate parameter names if possible (e.g. rgb = thisTrial_2.rgb)
if thisTrial_2 != None:
    for paramName in thisTrial_2:
        exec('{} = thisTrial_2[paramName]'.format(paramName))
```

```
for thisTrial_2 in trials:
    currentLoop = trials
    # abbreviate parameter names if possible (e.g. rgb = thisTrial_2.rgb)
    if thisTrial_2 != None:
        for paramName in thisTrial_2:
            exec('{} = thisTrial_2[paramName]'.format(paramName))

    # set up handler to look after randomisation of conditions etc
    thisTrial = data.TrialHandler(nReps=1.0, method='random',
        extraInfo=expInfo, originPath=-1,

trialList=data.importConditions('picture_setup_dot_probe_fnirs_eye_tracker.xlsx',
selection=list_random_blocks[random_blocks_counter]),
        seed=None, name='thisTrial')
    thisExp.addLoop(thisTrial)  # add the loop to the experiment
    thisThisTrial = thisTrial.trialList[0]  # so we can initialise stimuli with some values
    # abbreviate parameter names if possible (e.g. rgb = thisThisTrial.rgb)
    if thisThisTrial != None:
        for paramName in thisThisTrial:
            exec('{} = thisThisTrial[paramName]'.format(paramName))

    for thisThisTrial in thisTrial:
        currentLoop = thisTrial
        # abbreviate parameter names if possible (e.g. rgb = thisThisTrial.rgb)
        if thisThisTrial != None:
            for paramName in thisThisTrial:
```

```
        exec('{} = thisThisTrial[paramName]'.format(paramName))

    # --- Prepare to start Routine "Fixation_Cross" ---
    continueRoutine = True
    routineForceEnded = False
    # update component parameters for each repeat
    # Run 'Begin Routine' code from fixation_cross_code
    #Sending trail info to the eye tracker
    el_tracker.sendMessage('TRIALID %d' % trial_index)

    trial_index += 1

    #el_tracker.sendMessage(trial_initial_info)
    el_tracker.sendMessage('Fixation_Cross_Start')

    el_tracker.sendMessage('!V DRAWLINE 255 255 255 960 505 960 575')
    el_tracker.sendMessage('!V DRAWLINE 255 255 255 925 540 995 540')

    #Sending triggers at the start of the first trial in each block
    if trigger_sent == False:

        #Trigger for neutral/neutral face stiumlus
        if Face_Pairs == 'neutral/neutral':
            outlet.push_sample([2])

        #Trigger for happy/neutral congurent
        if Face_Pairs == 'happy/neutral' and Type == 'Congurent':
            outlet.push_sample([3])

        #Trigger for happy/neutral incongurent
        if Face_Pairs == 'happy/neutral' and Type == 'Incongurent':
            outlet.push_sample([4])

        #Trigger for fearful/neutral congurent
        if Face_Pairs == 'fearful/neutral' and Type == 'Congurent':
            outlet.push_sample([5])

        #Trigger for fearful/neutral incongurent
        if Face_Pairs == 'fearful/neutral' and Type == 'Incongurent':
            outlet.push_sample([6])

        trigger_sent = True





    # record trial variables to the EDF data file, for details, see Data
    # Viewer User Manual, "Protocol for EyeLink Data to Viewer Integration"
    #el_tracker.sendMessage('!V TRIAL_VAR Trial_Number %s' % Trial_Number)
    #el_tracker.sendMessage('!V TRIAL_VAR Participant_ID %s' % Participant_ID)
```

```
    #el_tracker.sendMessage('!V TRIAL_VAR Face_Stimuli_Left %s' %
Face_Stimuli_Left)
    #el_tracker.sendMessage('!V TRIAL_VAR Face_Stimuli_Right %s' %
Face_Stimuli_Right)
    #el_tracker.sendMessage('!V TRIAL_VAR Type %s' % Type)
    #el_tracker.sendMessage('!V TRIAL_VAR Face_Pairs %s' % Face_Pairs)
    #el_tracker.sendMessage('!V TRIAL_VAR Gender %s' % Gender)
    #el_tracker.sendMessage('!V TRIAL_VAR Dot_Location %s' % Dot_Location)
    #el_tracker.sendMessage('!V TRIAL_VAR Reaction_Time %.10f' %
Reaction_Time)
    #el_tracker.sendMessage('!V TRIAL_VAR Response_Accuracy %d' %
Response_Accuracy)
    # keep track of which components have finished
    Fixation_CrossComponents = [fixation_cross]
    for thisComponent in Fixation_CrossComponents:
        thisComponent.tStart = None
        thisComponent.tStop = None
        thisComponent.tStartRefresh = None
        thisComponent.tStopRefresh = None
        if hasattr(thisComponent, 'status'):
            thisComponent.status = NOT_STARTED
    # reset timers
    t = 0
    _timeToFirstFrame = win.getFutureFlipTime(clock="now")
    frameN = -1

    # --- Run Routine "Fixation_Cross" ---
    while continueRoutine:
        # get current time
        t = routineTimer.getTime()
        tThisFlip = win.getFutureFlipTime(clock=routineTimer)
        tThisFlipGlobal = win.getFutureFlipTime(clock=None)
        frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
        # update/draw components on each frame
        # Run 'Each Frame' code from fixation_cross_code
        if t > Time_Jitter:
            break

        # *fixation_cross* updates
        if fixation_cross.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
            # keep track of start time/frame for later
            fixation_cross.frameNStart = frameN  # exact frame index
            fixation_cross.tStart = t  # local t and not account for scr refresh
            fixation_cross.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(fixation_cross, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'fixation_cross.started')
            fixation_cross.setAutoDraw(True)

        # check for quit (typically the Esc key)
        if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
            core.quit()
        # check if all components have finished
        if not continueRoutine:  # a component has requested a forced-end of
Routine
            routineForceEnded = True
            break
        continueRoutine = False  # will revert to True if at least one component still
running
        for thisComponent in Fixation_CrossComponents:
            if hasattr(thisComponent, "status") and thisComponent.status !=
FINISHED:
                    continueRoutine = True
                    break  # at least one component has not yet finished

        # refresh the screen
        if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
            win.flip()

    # --- Ending Routine "Fixation_Cross" ---
    for thisComponent in Fixation_CrossComponents:
        if hasattr(thisComponent, "setAutoDraw"):
            thisComponent.setAutoDraw(False)
    # Run 'End Routine' code from fixation_cross_code
    el_tracker.sendMessage('Fixation_Cross_Stop')
    # the Routine "Fixation_Cross" was not non-slip safe, so reset the non-slip timer
    routineTimer.reset()
```

```
# --- Prepare to start Routine "Faces_Stimuli" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
# Run 'Begin Routine' code from code_faces_stimuli
#To find ratio from psychopy take use this formula:(scn_width/2)-(psychopy_width * scn_height)
left_image_center_x_axis = int((scn_width/2)-(687))
right_image_center_x_axis = int((scn_width/2)+(687))
image_center_y_axis = int(scn_height/2.0)
image_width = int(362)
image_height = int(506)

#'!V IMGLOAD CENTER %s %d %d %d %d' % (bg_image, int(scn_width/2.0), int(scn_height/2.0), int(scn_width), int(scn_height))
#!V IMGLOAD CENTER <relative_image_path> <x_position> <y_position> [width] [height]
image_face_left = "../../" + Face_Stimuli_Left
image_face_right = "../../" + Face_Stimuli_Right

el_tracker.sendMessage('!V IMGLOAD CENTER %s %d %d %d %d' % (image_face_left, left_image_center_x_axis, image_center_y_axis, image_width, image_height))
el_tracker.sendMessage('!V IMGLOAD CENTER %s %d %d %d %d' % (image_face_right, right_image_center_x_axis, image_center_y_axis, image_width, image_height))
```

```
        left_image_left_border = left_image_center_x_axis - image_width/2 #left left
        left_image_right_border= left_image_center_x_axis + image_width/2 #left right
        right_image_left_border= right_image_center_x_axis - image_width/2 #right left
        right_image_right_border= right_image_center_x_axis + image_width/2 #right
right
        top = image_center_y_axis + image_height/2  #top
        bottom= image_center_y_axis - image_height/2  #bottom

        # send interest area messages to record in the EDF data file
        # here we draw a rectangular IA, for illustration purposes
        # format: !V IAREA RECTANGLE <id> <left> <top> <right> <bottom> [label]
        # for all supported interest area commands, see the Data Viewer Manual,
        # "Protocol for EyeLink Data to Viewer Integration"
        ia_image_left = (1, left_image_left_border, top, left_image_right_border, bottom
, 'square')
        ia_image_right = (2, right_image_left_border, top, right_image_right_border,
bottom , 'square')

        el_tracker.sendMessage('!V IAREA RECTANGLE %d %d %d %d %d %s' %
ia_image_left)
        el_tracker.sendMessage('!V IAREA RECTANGLE %d %d %d %d %d %s' %
ia_image_right)



        el_tracker.sendMessage('Faces_Stimuli_Start')
        image_left.setImage(Face_Stimuli_Left)
        image_right.setImage(Face_Stimuli_Right)
```

```
# keep track of which components have finished
Faces_StimuliComponents = [image_left, image_right, fixation_cross_2]
for thisComponent in Faces_StimuliComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "Faces_Stimuli" ---
while continueRoutine and routineTimer.getTime() < 1.2:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # *image_left* updates
    if image_left.status == NOT_STARTED and tThisFlip >= 0-frameTolerance:
        # keep track of start time/frame for later
        image_left.frameNStart = frameN  # exact frame index
        image_left.tStart = t  # local t and not account for scr refresh
        image_left.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(image_left, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'image_left.started')
        image_left.setAutoDraw(True)
    if image_left.status == STARTED:
        # is it time to stop? (based on global clock, using actual start)
        if tThisFlipGlobal > image_left.tStartRefresh + 1.2-frameTolerance:
            # keep track of stop time/frame for later
            image_left.tStop = t  # not accounting for scr refresh
            image_left.frameNStop = frameN  # exact frame index
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'image_left.stopped')
            image_left.setAutoDraw(False)

    # *image_right* updates
    if image_right.status == NOT_STARTED and tThisFlip >= 0-frameTolerance:
        # keep track of start time/frame for later
        image_right.frameNStart = frameN  # exact frame index
        image_right.tStart = t  # local t and not account for scr refresh
        image_right.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(image_right, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'image_right.started')
```

```
            image_right.setAutoDraw(True)
        if image_right.status == STARTED:
            # is it time to stop? (based on global clock, using actual start)
            if tThisFlipGlobal > image_right.tStartRefresh + 1.2-frameTolerance:
                # keep track of stop time/frame for later
                image_right.tStop = t  # not accounting for scr refresh
                image_right.frameNStop = frameN  # exact frame index
                # add timestamp to datafile
                thisExp.timestampOnFlip(win, 'image_right.stopped')
                image_right.setAutoDraw(False)

        # *fixation_cross_2* updates
        if fixation_cross_2.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
            # keep track of start time/frame for later
            fixation_cross_2.frameNStart = frameN  # exact frame index
            fixation_cross_2.tStart = t  # local t and not account for scr refresh
            fixation_cross_2.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(fixation_cross_2, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'fixation_cross_2.started')
            fixation_cross_2.setAutoDraw(True)
        if fixation_cross_2.status == STARTED:
            # is it time to stop? (based on global clock, using actual start)
            if tThisFlipGlobal > fixation_cross_2.tStartRefresh + 1.2-frameTolerance:
                # keep track of stop time/frame for later
                fixation_cross_2.tStop = t  # not accounting for scr refresh
                fixation_cross_2.frameNStop = frameN  # exact frame index
                # add timestamp to datafile
                thisExp.timestampOnFlip(win, 'fixation_cross_2.stopped')
                fixation_cross_2.setAutoDraw(False)

        # check for quit (typically the Esc key)
        if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
            core.quit()

        # check if all components have finished
        if not continueRoutine:  # a component has requested a forced-end of
Routine
            routineForceEnded = True
            break
        continueRoutine = False  # will revert to True if at least one component still
running
        for thisComponent in Faces_StimuliComponents:
            if hasattr(thisComponent, "status") and thisComponent.status !=
FINISHED:
                continueRoutine = True
                break  # at least one component has not yet finished

        # refresh the screen
        if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
            win.flip()
        # --- Ending Routine "Faces_Stimuli" ---
        for thisComponent in Faces_StimuliComponents:
            if hasattr(thisComponent, "setAutoDraw"):
                thisComponent.setAutoDraw(False)
        # Run 'End Routine' code from code_faces_stimuli
        el_tracker.sendMessage('Faces_Stimuli_Stop')

        # Send a message to clear the Data Viewer screen
        bgcolor_RGB = (0, 0, 0)
        el_tracker.sendMessage('!V CLEAR %d %d %d' % bgcolor_RGB)



        # using non-slip timing so subtract the expected duration of this Routine (unless
ended on request)
        if routineForceEnded:
            routineTimer.reset()
        else:
            routineTimer.addTime(-1.200000)

        # --- Prepare to start Routine "Dot_Stimuli" ---
        continueRoutine = True
        routineForceEnded = False
        # update component parameters for each repeat
        # Run 'Begin Routine' code from code_dot_stimuli
        el_tracker.sendMessage('Dot_Stimuli_Start')
        if Dot_Location == 'left':
            show_dot_left = True

            #Drawing left dot to edf file
            el_tracker.sendMessage('!V DRAWLINE 255 255 255 %d %d %d %d' %
(left_image_center_x_axis, (image_center_y_axis - 20), left_image_center_x_axis,
(image_center_y_axis + 20)))
            el_tracker.sendMessage('!V DRAWLINE 255 255 255 %d %d %d %d' %
((left_image_center_x_axis - 20), image_center_y_axis, (left_image_center_x_axis +
20), image_center_y_axis))



        if Dot_Location == 'right':
            show_dot_right = True

            #Drawing right dot to edf
            el_tracker.sendMessage('!V DRAWLINE 255 255 255 %d %d %d %d' %
(right_image_center_x_axis, (image_center_y_axis - 20), right_image_center_x_axis,
(image_center_y_axis + 20)))
            el_tracker.sendMessage('!V DRAWLINE 255 255 255 %d %d %d %d' %
((right_image_center_x_axis - 20), image_center_y_axis,
(right_image_center_x_axis + 20), image_center_y_axis))

        key_reaction.keys = []
        key_reaction.rt = []
        _key_reaction_allKeys = []
        # keep track of which components have finished
```

```
Dot_StimuliComponents = [key_reaction, dot1_left, dot1_right, fixation_cross_3]
for thisComponent in Dot_StimuliComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "Dot_Stimuli" ---
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # *key_reaction* updates
    waitOnFlip = False
    if key_reaction.status == NOT_STARTED and tThisFlip >= 0-frameTolerance:
        # keep track of start time/frame for later
```

```
            key_reaction.frameNStart = frameN  # exact frame index
            key_reaction.tStart = t  # local t and not account for scr refresh
            key_reaction.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(key_reaction, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'key_reaction.started')
            key_reaction.status = STARTED
            # keyboard checking is just starting
            waitOnFlip = True
            win.callOnFlip(key_reaction.clock.reset)  # t=0 on next screen flip
            win.callOnFlip(key_reaction.clearEvents, eventType='keyboard')  # clear
events on next screen flip
        if key_reaction.status == STARTED and not waitOnFlip:
            theseKeys = key_reaction.getKeys(keyList=['1','2'], waitRelease=False)
            _key_reaction_allKeys.extend(theseKeys)
            if len(_key_reaction_allKeys):
                key_reaction.keys = _key_reaction_allKeys[0].name  # just the first key
pressed
                key_reaction.rt = _key_reaction_allKeys[0].rt
                # a response ends the routine
                continueRoutine = False

        # *dot1_left* updates
        if dot1_left.status == NOT_STARTED and show_dot_left == True:
            # keep track of start time/frame for later
            dot1_left.frameNStart = frameN  # exact frame index
            dot1_left.tStart = t  # local t and not account for scr refresh
            dot1_left.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(dot1_left, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'dot1_left.started')
            dot1_left.setAutoDraw(True)

        # *dot1_right* updates
        if dot1_right.status == NOT_STARTED and show_dot_right == True:
            # keep track of start time/frame for later
            dot1_right.frameNStart = frameN  # exact frame index
            dot1_right.tStart = t  # local t and not account for scr refresh
            dot1_right.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(dot1_right, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'dot1_right.started')
            dot1_right.setAutoDraw(True)

        # *fixation_cross_3* updates
        if fixation_cross_3.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
            # keep track of start time/frame for later
            fixation_cross_3.frameNStart = frameN  # exact frame index
            fixation_cross_3.tStart = t  # local t and not account for scr refresh
            fixation_cross_3.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(fixation_cross_3, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'fixation_cross_3.started')
            fixation_cross_3.setAutoDraw(True)

        # check for quit (typically the Esc key)
        if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
            core.quit()

        # check if all components have finished
        if not continueRoutine:  # a component has requested a forced-end of
Routine
            routineForceEnded = True
            break
        continueRoutine = False  # will revert to True if at least one component still
running
        for thisComponent in Dot_StimuliComponents:
            if hasattr(thisComponent, "status") and thisComponent.status !=
FINISHED:
                continueRoutine = True
                break  # at least one component has not yet finished

        # refresh the screen
        if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
            win.flip()

    # --- Ending Routine "Dot_Stimuli" ---
    for thisComponent in Dot_StimuliComponents:
        if hasattr(thisComponent, "setAutoDraw"):
            thisComponent.setAutoDraw(False)
    # Run 'End Routine' code from code_dot_stimuli
    el_tracker.sendMessage('Dot_Stimuli_Stop')

    #Getting time for when the button is pushed.
    Experiment_Time = expClock.getTime()

    #Getting the reactiontime for excel document
    Reaction_Time = key_reaction.rt

    #Check if participant pressed the right button (1 = right, 0 = wrong)
    if ((Dot_Location == 'left') and (key_reaction.keys == '1')) or ((Dot_Location ==
'right') and (key_reaction.keys == '2')):
        Response_Accuracy = 1
    else:
        Response_Accuracy = 0



    show_dot_left = False
    show_dot_right = False

    #Gets the highest reaction time
    if Response_Accuracy == 1 and (Reaction_Time < Reaction_Time_Record):
        Reaction_Time_Record = Reaction_Time
    #Putting all the info in the list

excel_list.append([Trial_Number,Participant_ID,Face_Stimuli_Left,Face_Stimuli_Rig
ht,Type,Face_Pairs,Gender,Dot_Location,Response_Accuracy,Reaction_Time,Expe
riment_Time, Time_Jitter, VAS])

    # check responses
    if key_reaction.keys in ['', [], None]:  # No response was made
        key_reaction.keys = None
    thisTrial.addData('key_reaction.keys',key_reaction.keys)
    if key_reaction.keys != None:  # we had a response
        thisTrial.addData('key_reaction.rt', key_reaction.rt)
    # the Routine "Dot_Stimuli" was not non-slip safe, so reset the non-slip timer
    routineTimer.reset()

    # --- Prepare to start Routine "Rest" ---
    continueRoutine = True
    routineForceEnded = False
    # update component parameters for each repeat
    # Run 'Begin Routine' code from code_rest
    el_tracker.sendMessage('Pause_Start')



    #Making a list for eye tracking messages (used to send messages at a slower
paste)
    et_message_list = []
    et_message_list.append('!V TRIAL_VAR Participant_ID %s' % Participant_ID)
    et_message_list.append('!V TRIAL_VAR Trial_Number %s' % Trial_Number)
    et_message_list.append('!V TRIAL_VAR Face_Stimuli_Left %s' %
Face_Stimuli_Left)
    et_message_list.append('!V TRIAL_VAR Face_Stimuli_Right %s' %
Face_Stimuli_Right)
    et_message_list.append('!V TRIAL_VAR Type %s' % Type)
    et_message_list.append('!V TRIAL_VAR Face_Pairs %s' % Face_Pairs)
    et_message_list.append('!V TRIAL_VAR Gender %s' % Gender)
    et_message_list.append('!V TRIAL_VAR Dot_Location %s' % Dot_Location)
    et_message_list.append('!V TRIAL_VAR Time_Jitter %d' % Time_Jitter)
    et_message_list.append('!V TRIAL_VAR Emotion_Side %s' % Emotion_Side)
    et_message_list.append('!V TRIAL_VAR Reaction_Time %.10f' %
Reaction_Time)
    et_message_list.append('!V TRIAL_VAR Response_Accuracy %d' %
Response_Accuracy)



    t2 = 0.1
    counter = 0

    # keep track of which components have finished
    RestComponents = [reaction_time_text]
    for thisComponent in RestComponents:
        thisComponent.tStart = None
        thisComponent.tStop = None
        thisComponent.tStartRefresh = None
```

```
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "Rest" ---
while continueRoutine and routineTimer.getTime() < 1.5:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame
    # Run 'Each Frame' code from code_rest
    #Timer to send eye tracking messages at a slower paste
    t1 = t

    if t1 > t2 and counter < 10:
        el_tracker.sendMessage(et_message_list[counter])
        t2 = t + 0.1
        counter += 1
        # *reaction_time_text* updates
        if reaction_time_text.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
            # keep track of start time/frame for later
            reaction_time_text.frameNStart = frameN  # exact frame index
            reaction_time_text.tStart = t  # local t and not account for scr refresh
            reaction_time_text.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(reaction_time_text, 'tStartRefresh')  # time at next scr
refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'reaction_time_text.started')
            reaction_time_text.setAutoDraw(True)
        if reaction_time_text.status == STARTED:
            # is it time to stop? (based on global clock, using actual start)
            if tThisFlipGlobal > reaction_time_text.tStartRefresh + 1.5-frameTolerance:
                # keep track of stop time/frame for later
                reaction_time_text.tStop = t  # not accounting for scr refresh
                reaction_time_text.frameNStop = frameN  # exact frame index
                # add timestamp to datafile
                thisExp.timestampOnFlip(win, 'reaction_time_text.stopped')
                reaction_time_text.setAutoDraw(False)

        # check for quit (typically the Esc key)
        if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
            core.quit()
        # check if all components have finished
        if not continueRoutine:  # a component has requested a forced-end of
Routine
            routineForceEnded = True
            break
        continueRoutine = False  # will revert to True if at least one component still
running
        for thisComponent in RestComponents:
            if hasattr(thisComponent, "status") and thisComponent.status !=
FINISHED:
                continueRoutine = True
                break  # at least one component has not yet finished

        # refresh the screen
        if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
            win.flip()

    # --- Ending Routine "Rest" ---
    for thisComponent in RestComponents:
        if hasattr(thisComponent, "setAutoDraw"):
            thisComponent.setAutoDraw(False)
    # Run 'End Routine' code from code_rest
    el_tracker.sendMessage('Pause_Stop')

    #End trial for eye-tracker
    el_tracker.sendMessage('TRIAL_RESULT %d' % pylink.TRIAL_OK)
    # using non-slip timing so subtract the expected duration of this Routine (unless
ended on request)
    if routineForceEnded:
        routineTimer.reset()
    else:
        routineTimer.addTime(-1.500000)
    thisExp.nextEntry()

# completed 1.0 repeats of 'thisTrial'

# get names of stimulus parameters
if thisTrial.trialList in ([], [None], None):
    params = []
else:
    params = thisTrial.trialList[0].keys()
# save data for this loop
thisTrial.saveAsExcel(filename + '.xlsx', sheetName='thisTrial',
    stimOut=params,
    dataOut=['n','all_mean','all_std', 'all_raw'])
thisTrial.saveAsText(filename + 'thisTrial.csv', delim=',',
    stimOut=params,
    dataOut=['n','all_mean','all_std', 'all_raw'])

# --- Prepare to start Routine "inter_trial_interval" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
# Run 'Begin Routine' code from trial_end_trigger_code
#Trigger for rest start
outlet.push_sample([7])

#Go to next block
random_blocks_counter += 1

#Resets trigger variable
trigger_sent = False

#Removes decimals from reaction time to display
Reaction_Time_Display = str(Reaction_Time_Record)[:-11]
reaction_time_numbers_2.setText(Reaction_Time_Display)
# keep track of which components have finished
inter_trial_intervalComponents = [text, reaction_time_text_2,
reaction_time_numbers_2]
for thisComponent in inter_trial_intervalComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
```

```
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "inter_trial_interval" ---
while continueRoutine and routineTimer.getTime() < 15.0:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # *text* updates
    if text.status == NOT_STARTED and tThisFlip >= 0.0-frameTolerance:
        # keep track of start time/frame for later
        text.frameNStart = frameN  # exact frame index
        text.tStart = t  # local t and not account for scr refresh
        text.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(text, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'text.started')
        text.setAutoDraw(True)
    if text.status == STARTED:
        # is it time to stop? (based on global clock, using actual start)
        if tThisFlipGlobal > text.tStartRefresh + 15-frameTolerance:
            # keep track of stop time/frame for later
            text.tStop = t  # not accounting for scr refresh
            text.frameNStop = frameN  # exact frame index
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'text.stopped')
            text.setAutoDraw(False)

    # *reaction_time_text_2* updates
    if reaction_time_text_2.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
        # keep track of start time/frame for later
        reaction_time_text_2.frameNStart = frameN  # exact frame index
        reaction_time_text_2.tStart = t  # local t and not account for scr refresh
        reaction_time_text_2.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(reaction_time_text_2, 'tStartRefresh')  # time at next scr
refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'reaction_time_text_2.started')
        reaction_time_text_2.setAutoDraw(True)
    if reaction_time_text_2.status == STARTED:
        # is it time to stop? (based on global clock, using actual start)
        if tThisFlipGlobal > reaction_time_text_2.tStartRefresh + 2.5-frameTolerance:
            # keep track of stop time/frame for later
            reaction_time_text_2.tStop = t  # not accounting for scr refresh
            reaction_time_text_2.frameNStop = frameN  # exact frame index
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'reaction_time_text_2.stopped')
            reaction_time_text_2.setAutoDraw(False)

    # *reaction_time_numbers_2* updates
    if reaction_time_numbers_2.status == NOT_STARTED and tThisFlip >= 0.0-frameTolerance:
        # keep track of start time/frame for later
        reaction_time_numbers_2.frameNStart = frameN  # exact frame index
        reaction_time_numbers_2.tStart = t  # local t and not account for scr refresh
        reaction_time_numbers_2.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(reaction_time_numbers_2, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'reaction_time_numbers_2.started')
        reaction_time_numbers_2.setAutoDraw(True)
    if reaction_time_numbers_2.status == STARTED:
        # is it time to stop? (based on global clock, using actual start)
        if tThisFlipGlobal > reaction_time_numbers_2.tStartRefresh + 2.5-frameTolerance:
            # keep track of stop time/frame for later
            reaction_time_numbers_2.tStop = t  # not accounting for scr refresh
            reaction_time_numbers_2.frameNStop = frameN  # exact frame index
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'reaction_time_numbers_2.stopped')
            reaction_time_numbers_2.setAutoDraw(False)

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
```

```
            core.quit()

        # check if all components have finished
        if not continueRoutine:  # a component has requested a forced-end of Routine
            routineForceEnded = True
            break
        continueRoutine = False  # will revert to True if at least one component still
running
        for thisComponent in inter_trial_intervalComponents:
            if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
                continueRoutine = True
                break  # at least one component has not yet finished

        # refresh the screen
        if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
            win.flip()

    # --- Ending Routine "inter_trial_interval" ---
    for thisComponent in inter_trial_intervalComponents:
        if hasattr(thisComponent, "setAutoDraw"):
            thisComponent.setAutoDraw(False)
    # using non-slip timing so subtract the expected duration of this Routine (unless
ended on request)
    if routineForceEnded:
        routineTimer.reset()
    else:
        routineTimer.addTime(-15.000000)
    thisExp.nextEntry()

# completed 20.0 repeats of 'trials'


# get names of stimulus parameters
if trials.trialList in ([], [None], None):
    params = []
else:
    params = trials.trialList[0].keys()
# save data for this loop
trials.saveAsExcel(filename + '.xlsx', sheetName='trials',
    stimOut=params,
    dataOut=['n','all_mean','all_std', 'all_raw'])
trials.saveAsText(filename + 'trials.csv', delim=',',
    stimOut=params,
    dataOut=['n','all_mean','all_std', 'all_raw'])

# --- Prepare to start Routine "experiment_ended" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
# Run 'Begin Routine' code from end_dot_probe_code
#Trigger for experiment ended
outlet.push_sample([8])
```

```
#Message to eyetracker
el_tracker.sendMessage('endExperiment')



# keep track of which components have finished
experiment_endedComponents = [dot_probe_ended_text]
for thisComponent in experiment_endedComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "experiment_ended" ---
while continueRoutine and routineTimer.getTime() < 5.0:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame
    # *dot_probe_ended_text* updates
    if dot_probe_ended_text.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
        # keep track of start time/frame for later
        dot_probe_ended_text.frameNStart = frameN  # exact frame index
        dot_probe_ended_text.tStart = t  # local t and not account for scr refresh
        dot_probe_ended_text.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(dot_probe_ended_text, 'tStartRefresh')  # time at next scr
refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'dot_probe_ended_text.started')
        dot_probe_ended_text.setAutoDraw(True)
    if dot_probe_ended_text.status == STARTED:
        # is it time to stop? (based on global clock, using actual start)
        if tThisFlipGlobal > dot_probe_ended_text.tStartRefresh + 5-frameTolerance:
            # keep track of stop time/frame for later
            dot_probe_ended_text.tStop = t  # not accounting for scr refresh
            dot_probe_ended_text.frameNStop = frameN  # exact frame index
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'dot_probe_ended_text.stopped')
            dot_probe_ended_text.setAutoDraw(False)

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()
    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in experiment_endedComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished

    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()

# --- Ending Routine "experiment_ended" ---
for thisComponent in experiment_endedComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
# Run 'End Routine' code from end_dot_probe_code
#Saves the info from dot-probe experment list in a excel document
np.savetxt(Participant_ID + "_" + expName + "_" + expInfo['date'] + ".csv", excel_list,
delimiter = ",", fmt ='% s')

el_tracker.stopRecording()
```

```
# using non-slip timing so subtract the expected duration of this Routine (unless
ended on request)
if routineForceEnded:
    routineTimer.reset()
else:
    routineTimer.addTime(-5.000000)
# Run 'End Experiment' code from Start_and_end_code
# Step 7: disconnect, download the EDF file, then terminate the task
terminate_task()

# --- End experiment ---
# Flip one final time so any remaining win.callOnFlip()
# and win.timeOnFlip() tasks get executed before quitting
win.flip()

# these shouldn't be strictly necessary (should auto-save)
thisExp.saveAsPickle(filename)
logging.flush()
# make sure everything is closed down
if eyetracker:
    eyetracker.setConnectionState(False)
thisExp.abort()  # or data files will save again on exit
win.close()
core.quit()
```

A.4 EEG/ET PSYCHOPY CODE

Note that this one also contains a resting state task and a few visual search tasks.

```
# --- Import packages ---
from psychopy import locale_setup
from psychopy import prefs
from psychopy import sound, gui, visual, core, data, event, logging, clock, colors,
layout, parallel
from psychopy.constants import (NOT_STARTED, STARTED, PLAYING, PAUSED,
                                STOPPED, FINISHED, PRESSED, RELEASED, FOREVER)

import numpy as np  # whole numpy lib is available, prepend 'np.'
from numpy import (sin, cos, tan, log, log10, pi, average,
                   sqrt, std, deg2rad, rad2deg, linspace, asarray)
from numpy.random import random, randint, normal, shuffle, choice as randchoice
import os  # handy system and path functions
import sys  # to get file system encoding

import psychopy.iohub as io
from psychopy.hardware import keyboard

# Run 'Before Experiment' code from Start_and_end_code
import __future__
import pylink
import os
import platform
import random

import time

import sys

from EyeLinkCoreGraphicsPsychoPy import EyeLinkCoreGraphicsPsychoPy

from psychopy import visual, core, event, monitors, gui

from PIL import Image  # for preparing the Host backdrop image

from string import ascii_letters, digits



#Setting up gloabl variable for reaction time and vas-scale

global Reaction_Time_Display

global Reaction_Time

global Response_Accuracy

global VAS



# Switch to the script folder

script_path = os.path.dirname(sys.argv[0])

if len(script_path) != 0:

    os.chdir(script_path)


# Show only critical log message in the PsychoPy console

from psychopy import logging

logging.console.setLevel(logging.CRITICAL)


#List to later be stored in excel document (this is the heading)
```

```
excel_list =
[['Trial_Number','Participant_ID','Face_Stimuli_Left','Face_Stimuli_Right','Type','Face
_Pairs','Gender','Dot_Location','Response_Accuracy','Reaction_Time','Experiemnt_Ti
me', 'Time_Jitter', 'VAS']]


#Variable  that is later combined with timer to decide when and what dot will show
show_dot_left = False
show_dot_right = False


Trial_Number = 0


# Set this variable to True if you use the built-in retina screen as your
# primary display device on macOS. If have an external monitor, set this
# variable True if you choose to "Optimize for Built-in Retina Display"
# in the Displays preference settings.
use_retina = False

# Set this variable to True to run the script in "Dummy Mode"
dummy_mode = False

# Set this variable to True to run the task in full screen mode
# It is easier to debug the script in non-fullscreen mode
full_screen = True
```

```
# Set up EDF data file name and local data folder
#
# The EDF data filename should not exceed 8 alphanumeric characters
# use ONLY number 0-9, letters, & _ (underscore) in the filename
edf_fname = ''

# Prompt user to specify an EDF data filename
# before we open a fullscreen window
dlg_title = 'Enter ParticipantID'
dlg_prompt = 'Please enter a file name with 8 or fewer characters\n' + \
             '[letters, numbers, and underscore].'

# loop until we get a valid filename
while True:
    dlg = gui.Dlg(dlg_title)
    dlg.addText(dlg_prompt)
    dlg.addField('ParticipantID:', edf_fname)
    # show dialog and wait for OK or Cancel
    ok_data = dlg.show()
    if dlg.OK:  # if ok_data is not None
        print('EDF data filename: {}'.format(ok_data[0]))
    else:
        print('user cancelled')
        core.quit()
        sys.exit()
    # get the string entered by the experimenter
    tmp_str = dlg.data[0]
    # strip trailing characters, ignore the ".edf" extension
    edf_fname = tmp_str.rstrip().split('.')[0]

    # check if the filename is valid (length <= 8 & no special char)
    allowed_char = ascii_letters + digits + '_'
    if not all([c in allowed_char for c in edf_fname]):
        print('ERROR: Invalid EDF filename')
    elif len(edf_fname) > 8:
        print('ERROR: EDF filename should not exceed 8 characters')
    else:
        break


# Set up a folder to store the EDF data files and the associated resources
# e.g., files defining the interest areas used in each trial
results_folder = 'Eye_EEG'
if not os.path.exists(results_folder):
    os.makedirs(results_folder)

# We download EDF data file from the EyeLink Host PC to the local hard
# drive at the end of each testing session, here we rename the EDF to
# include session start date/time
time_str = time.strftime("_%Y_%m_%d_%H_%M", time.localtime())
```

```
session_identifier = edf_fname + "_EEG" + time_str


# create a folder for the current testing session in the "results" folder
session_folder = os.path.join(results_folder, session_identifier)
if not os.path.exists(session_folder):
    os.makedirs(session_folder)


# Step 1: Connect to the EyeLink Host PC
#
# The Host IP address, by default, is "100.1.1.1".
# the "el_tracker" objected created here can be accessed through the Pylink
# Set the Host PC address to "None" (without quotes) to run the script
# in "Dummy Mode"
if dummy_mode:
    el_tracker = pylink.EyeLink(None)
else:
    try:
        el_tracker = pylink.EyeLink("100.1.1.1")
    except RuntimeError as error:
        print('ERROR:', error)
        core.quit()
        sys.exit()


# Step 2: Open an EDF data file on the Host PC
edf_file = edf_fname + "_EEG"+ ".EDF"
```

```
try:
    el_tracker.openDataFile(edf_file)
except RuntimeError as err:
    print('ERROR:', err)
    # close the link if we have one open
    if el_tracker.isConnected():
        el_tracker.close()
    core.quit()
    sys.exit()

# Add a header text to the EDF file to identify the current experiment name
# This is OPTIONAL. If your text starts with "RECORDED BY " it will be
# available in DataViewer's Inspector window by clicking
# the EDF session node in the top panel and looking for the "Recorded By:"
# field in the bottom panel of the Inspector.
preamble_text = 'RECORDED BY %s' % os.path.basename(__file__)
el_tracker.sendCommand("add_file_preamble_text '%s'" % preamble_text)

# Step 3: Configure the tracker
#
# Put the tracker in offline mode before we change tracking parameters
el_tracker.setOfflineMode()

# Get the software version:  1-EyeLink I, 2-EyeLink II, 3/4-EyeLink 1000,
# 5-EyeLink 1000 Plus, 6-Portable DUO
```

```
eyelink_ver = 0  # set version to 0, in case running in Dummy mode
if not dummy_mode:
    vstr = el_tracker.getTrackerVersionString()
    eyelink_ver = int(vstr.split()[-1].split('.')[0])
    # print out some version info in the shell
    print('Running experiment on %s, version %d' % (vstr, eyelink_ver))

# File and Link data control
# what eye events to save in the EDF file, include everything by default
file_event_flags =
'LEFT,RIGHT,FIXATION,SACCADE,BLINK,MESSAGE,BUTTON,INPUT'
# what eye events to make available over the link, include everything by default
link_event_flags =
'LEFT,RIGHT,FIXATION,SACCADE,BLINK,BUTTON,FIXUPDATE,INPUT'
# what sample data to save in the EDF data file and to make available
# over the link, include the 'HTARGET' flag to save head target sticker
# data for supported eye trackers
if eyelink_ver > 3:
    file_sample_flags =
'LEFT,RIGHT,GAZE,HREF,RAW,AREA,HTARGET,GAZERES,BUTTON,STATUS,IN
PUT'
    link_sample_flags =
'LEFT,RIGHT,GAZE,GAZERES,AREA,HTARGET,STATUS,INPUT'
else:
    file_sample_flags =
'LEFT,RIGHT,GAZE,HREF,RAW,AREA,GAZERES,BUTTON,STATUS,INPUT'
    link_sample_flags = 'LEFT,RIGHT,GAZE,GAZERES,AREA,STATUS,INPUT'
```

```
el_tracker.sendCommand("file_event_filter = %s" % file_event_flags)

el_tracker.sendCommand("file_sample_data = %s" % file_sample_flags)

el_tracker.sendCommand("link_event_filter = %s" % link_event_flags)

el_tracker.sendCommand("link_sample_data = %s" % link_sample_flags)


# Optional tracking parameters

# Sample rate, 250, 500, 1000, or 2000, check your tracker specification

# if eyelink_ver > 2:

#     el_tracker.sendCommand("sample_rate 1000")

# Choose a calibration type, H3, HV3, HV5, HV13 (HV = horizontal/vertical),

el_tracker.sendCommand("calibration_type = HV9")

# Set a gamepad button to accept calibration/drift check target

# You need a supported gamepad/button box that is connected to the Host PC

el_tracker.sendCommand("button_function 5 'accept_target_fixation'")


# Step 4: set up a graphics environment for calibration

#

# Open a window, be sure to specify monitor parameters

#mon = monitors.Monitor('myMonitor', width=53.0, distance=115.0)

win = visual.Window(size=(1920,1080),

                    pos=(1920,0),

                    winType='pyglet',

                    units='pix')


# get the native screen resolution used by PsychoPy

scn_width, scn_height = win.size
```

```
# resolution fix for Mac retina displays
if 'Darwin' in platform.system():
    if use_retina:
        scn_width = int(scn_width/2.0)
        scn_height = int(scn_height/2.0)

# Pass the display pixel coordinates (left, top, right, bottom) to the tracker
# see the EyeLink Installation Guide, "Customizing Screen Settings"
el_coords = "screen_pixel_coords = 0 0 %d %d" % (scn_width - 1, scn_height - 1)
el_tracker.sendCommand(el_coords)

# Write a DISPLAY_COORDS message to the EDF file
# Data Viewer needs this piece of info for proper visualization, see Data
# Viewer User Manual, "Protocol for EyeLink Data to Viewer Integration"
dv_coords = "DISPLAY_COORDS  0 0 %d %d" % (scn_width - 1, scn_height - 1)
el_tracker.sendMessage(dv_coords)

# Configure a graphics environment (genv) for tracker calibration
genv = EyeLinkCoreGraphicsPsychoPy(el_tracker, win)
print(genv)  # print out the version number of the CoreGraphics library

# Set background and foreground colors for the calibration target
# in PsychoPy, (-1, -1, -1)=black, (1, 1, 1)=white, (0, 0, 0)=mid-gray
foreground_color = (-1, -1, -1)
background_color = win.color
genv.setCalibrationColors(foreground_color, background_color)
```

```
# Set up the calibration target
#
# The target could be a "circle" (default), a "picture", a "movie" clip,
# or a rotating "spiral". To configure the type of calibration target, set
# genv.setTargetType to "circle", "picture", "movie", or "spiral", e.g.,
# genv.setTargetType('picture')
#
# Use gen.setPictureTarget() to set a "picture" target
# genv.setPictureTarget(os.path.join('images', 'fixTarget.bmp'))
#
# Use genv.setMovieTarget() to set a "movie" target
# genv.setMovieTarget(os.path.join('videos', 'calibVid.mov'))

# Use a picture as the calibration target
genv.setTargetType('picture')
genv.setPictureTarget(os.path.join('images', 'fixTarget.bmp'))

# Configure the size of the calibration target (in pixels)
# this option applies only to "circle" and "spiral" targets
# genv.setTargetSize(24)

# Beeps to play during calibration, validation and drift correction
# parameters: target, good, error
#     target -- sound to play when target moves
#     good -- sound to play on successful operation
#     error -- sound to play on failure or interruption
# Each parameter could be ''--default sound, 'off'--no sound, or a wav file
genv.setCalibrationSounds('', '', '')

# resolution fix for macOS retina display issues
if use_retina:
    genv.fixMacRetinaDisplay()

# Request Pylink to use the PsychoPy window we opened above for calibration
pylink.openGraphicsEx(genv)



# define a few helper functions for trial handling



def clear_screen(win):
    """ clear up the PsychoPy window"""

    win.fillColor = genv.getBackgroundColor()
    win.flip()



def show_msg(win, text, wait_for_keypress=True):
    """ Show task instructions on screen"""

    msg = visual.TextStim(win, text,
                           color=genv.getForegroundColor(),
                           wrapWidth=scn_width/2)
    clear_screen(win)
    msg.draw()
    win.flip()

    # wait indefinitely, terminates upon any key press
    if wait_for_keypress:
        event.waitKeys()
        clear_screen(win)


def terminate_task():
    """ Terminate the task gracefully and retrieve the EDF data file

    file_to_retrieve: The EDF on the Host that we would like to download
    win: the current window used by the experimental script
    """

    el_tracker = pylink.getEYELINK()

    if el_tracker.isConnected():
        # Terminate the current trial first if the task terminated prematurely
        error = el_tracker.isRecording()
        if error == pylink.TRIAL_OK:
            abort_trial()
```

```
# Put tracker in Offline mode
el_tracker.setOfflineMode()

# Clear the Host PC screen and wait for 500 ms
el_tracker.sendCommand('clear_screen 0')
pylink.msecDelay(500)

# Close the edf data file on the Host
el_tracker.closeDataFile()

# Show a file transfer message on the screen
msg = 'EDF data is transferring from EyeLink Host PC...'
show_msg(win, msg, wait_for_keypress=False)

# Download the EDF data file from the Host PC to a local data folder
# parameters: source_file_on_the_host, destination_file_on_local_drive
local_edf = os.path.join(session_folder, session_identifier + '.EDF')
try:
    el_tracker.receiveDataFile(edf_file, local_edf)
except RuntimeError as error:
    print('ERROR:', error)

# Close the link to the tracker.
el_tracker.close()
```

```
    # close the PsychoPy window
    win.close()

    # quit PsychoPy
    core.quit()
    sys.exit()


def abort_trial():
    """Ends recording """

    el_tracker = pylink.getEYELINK()

    # Stop recording
    if el_tracker.isRecording():
        # add 100 ms to catch final trial events
        pylink.pumpDelay(100)
        el_tracker.stopRecording()

    # clear the screen
    clear_screen(win)
    # Send a message to clear the Data Viewer screen
    bgcolor_RGB = (116, 116, 116)
    el_tracker.sendMessage('!V CLEAR %d %d %d' % bgcolor_RGB)

    # send a message to mark trial end
    el_tracker.sendMessage('TRIAL_RESULT %d' % pylink.TRIAL_ERROR)

    return pylink.TRIAL_ERROR

visual.Window(size=(1920,1080),pos=(1920,0),screen=2)

# Step 5: Set up the camera and calibrate the tracker


# Show the task instructions
task_msg = 'Ready for eye tracker calibration\n'
if dummy_mode:
    task_msg = task_msg + '\nNow, press ENTER to start the task'
else:
    task_msg = task_msg + '\nNow, press ENTER twice to calibrate tracker'
show_msg(win, task_msg)

# skip this step if running the script in Dummy Mode
if not dummy_mode:
    try:
        el_tracker.doTrackerSetup()
    except RuntimeError as err:
        print('ERROR:', err)
        el_tracker.exitCalibration()
```

```
# Ensure that relative paths start from the same directory as this script
_thisDir = os.path.dirname(os.path.abspath(__file__))
os.chdir(_thisDir)
# Store info about the experiment session
psychopyVersion = '2022.2.4'
expName = 'EEG_DP'  # from the Builder filename that created this script
expInfo = {}
# --- Show participant info dialog --
dlg = gui.DlgFromDict(dictionary=expInfo, sortKeys=False, title=expName)
if dlg.OK == False:
    core.quit()  # user pressed cancel
expInfo['date'] = data.getDateStr()  # add a simple timestamp
expInfo['expName'] = expName
expInfo['psychopyVersion'] = psychopyVersion

# Data file name stem = absolute path + name; later add .psyexp, .csv, .log, etc
filename = _thisDir + os.sep + u'data/%s_%s_%s' % (edf_fname, expName,
expInfo['date'])

# An ExperimentHandler isn't essential but helps with data saving
thisExp = data.ExperimentHandler(name=expName, version='',
    extraInfo=expInfo, runtimeInfo=None,
    originPath='C:\\Users\\Sven Ivar Ougendal\\OneDrive - OsloMet\\Master - Brain
Health projects - Sandra Klonteig et al. - Sven Master Thesis\\3 - Experiment
paradigm\\EEG_Data_Psychopy\\EEG_Psychopy.py',
    savePickle=True, saveWideText=False,
    dataFileName=filename)
```

```
# save a log file for detail verbose info
logFile = logging.LogFile(filename+'.log', level=logging.EXP)
logging.console.setLevel(logging.WARNING)  # this outputs to the screen, not a file

endExpNow = False  # flag for 'escape' or other condition => quit the exp
frameTolerance = 0.001  # how close to onset before 'same' frame

# Start Code - component code to be run after the window creation

# --- Setup the Window ---
win = visual.Window(
    size=[1920, 1080], fullscr=True, screen=1,
    winType='pyglet', allowStencil=False,
    monitor='testMonitor', color='black', colorSpace='rgb',
    blendMode='avg', useFBO=True,
    units='height')
win.mouseVisible = False
# store frame rate of monitor if we can measure it
expInfo['frameRate'] = win.getActualFrameRate()
if expInfo['frameRate'] != None:
    frameDur = 1.0 / round(expInfo['frameRate'])
else:
    frameDur = 1.0 / 60.0  # could not measure, so guess
# --- Setup input devices ---
ioConfig = {}
```

```
# Setup iohub keyboard
ioConfig['Keyboard'] = dict(use_keymap='psychopy')


ioSession = '1'
if 'session' in expInfo:
    ioSession = str(expInfo['session'])
ioServer = io.launchHubServer(window=win, **ioConfig)
eyetracker = None


# create a default keyboard (e.g. to check for escape)
defaultKeyboard = keyboard.Keyboard(backend='iohub')


# --- Initialize components for Routine "start_and_eyetracking_calibration" ---
# Run 'Begin Experiment' code from Start_and_end_code
#Fetching ID for the excel document
Participant_ID = edf_fname


#Functin to map cordianates from psychopy
def map_cord(x,y):
    return (x-scn_width/2, y-scn_height/2)






# --- Initialize components for Routine "vas_measure" ---
```

```
vas_text = visual.TextStim(win=win, name='vas_text',
    text='From a scale from 0 - 9, how tired are you?\n(Higher number = more tired)',
    font='Open Sans',
    pos=(0, 0), height=0.05, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=-1.0);
vas_response = keyboard.Keyboard()

# --- Initialize components for Routine "start_eeg_reminder" ---
reminder_eeg_data_collection_text = visual.TextStim(win=win,
name='reminder_eeg_data_collection_text',
    text='Researcher will prepare recordings\n\nThe experiment will start soon',
    font='Open Sans',
    pos=(0, 0), height=0.05, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=0.0);
eeg_data_collection_on_key = keyboard.Keyboard()

# --- Initialize components for Routine "rest_state_OPEN_eye_info" ---
rest_state_open_text = visual.TextStim(win=win, name='rest_state_open_text',
    text="You are now going to go through a number of tasks.\n\nThe first task is to sitt
still and look at a fixation cross.\nThe task will take 2,5 min\n\nYou will get 15
secounds before the task to place your hands comfortably.\n\nPlease relax and focus
on the cross.\n\nPress 'L' or 'R' when you are ready.\n\n",
    font='Open Sans',
    pos=(0, 0), height=0.04, wrapWidth=None, ori=0.0, 
    color='white', colorSpace='rgb', opacity=None, 
    languageStyle='LTR',
    depth=0.0);
rest_state_open_key = keyboard.Keyboard()

# --- Initialize components for Routine "rest_state_OPEN_eye_start_counter" ---

# --- Initialize components for Routine "rest_state_OPEN_eye_TASK" ---

# --- Initialize components for Routine "rest_state_CLOSED_eye_info" ---
rest_state_closed_text = visual.TextStim(win=win, name='rest_state_closed_text',
    text="You are now going to go through a number of task.\n\nThe first task is to sitt still and close your eyes\nThe task will take 2,5 min\n\nYou will get 15 secounds before the task to place your hands comfortably.\n\nYou will hear a 'beep' sound when the task is finished \n\nPlease relax and close your eyes.\n\nPress 'L' or 'R' when you are ready.\n\n",
    font='Open Sans',
    pos=(0, 0), height=0.04, wrapWidth=None, ori=0.0, 
    color='white', colorSpace='rgb', opacity=None, 
    languageStyle='LTR',
    depth=0.0);
rest_state_closed_key = keyboard.Keyboard()

# --- Initialize components for Routine "rest_state_CLOSED_eye_start_counter" ---

# --- Initialize components for Routine "rest_state_CLOSED_eye_TASK" ---
# --- Initialize components for Routine "dot_probe_task_info" ---
task_info_text = visual.TextStim(win=win, name='task_info_text',
    text="The next task goes as follows:\n1. A fixation cross will apear, followed by two pictures on each side of the screen. \n2. After a short time, the pictures will disapear and a DOT will apear on either side of the screen. \n3. You objective is to press 'L' or 'R' as fast a possible when the DOT apears. \n4. Your reaction time will apear on the screen.\n5. Then the task repeats itself. It will take aproximatly 10min\n6. You will then get informaition on the next task\n\nTo get ready, but your left indexfinger on 'L' and right indexfinger on 'R'\nPress 'L' or 'R' to start the task",
    font='Open Sans',
    pos=(0, 0), height=0.04, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=0.0);
task_info_response = keyboard.Keyboard()

# --- Initialize components for Routine "dot_probe_start_counter" ---
Start_experiment = parallel.ParallelPort(address='0x3FF8')
dot_probe_text_counter = visual.TextStim(win=win, name='dot_probe_text_counter',
    text='',
    font='Open Sans',
    pos=(0, -0.1), height=0.1, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=-1.0);
dot_probe_start_text = visual.TextStim(win=win, name='dot_probe_start_text',
    text='The test starts in:',
    font='Open Sans',
    pos=(0, 0.1), height=0.1, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=-2.0);

# --- Initialize components for Routine "Fixation_Cross" ---
fixation_cross = visual.ShapeStim(
    win=win, name='fixation_cross', vertices='cross',units='pix',
    size=(35, 35),
    ori=0.0, pos=(0, 0), anchor='center',
    lineWidth=0.02,     colorSpace='rgb',  lineColor='white', fillColor='white',
    opacity=None, depth=-1.0, interpolate=True)
fixation = parallel.ParallelPort(address='0x3FF8')

# --- Initialize components for Routine "Faces_Stimuli" ---
image_left = visual.ImageStim(
    win=win,
    name='image_left', units='pix',
    image='sin', mask=None, anchor='center',
    ori=0.0, pos=(-687, 0), size=(362, 506),
    color=[1,1,1], colorSpace='rgb', opacity=None,
    flipHoriz=False, flipVert=False,
    texRes=128.0, interpolate=True, depth=-1.0)
image_right = visual.ImageStim(
    win=win,
    name='image_right', units='pix',
    image='sin', mask=None, anchor='center',
    ori=0.0, pos=(687, 0), size=(362, 506),
    color=[1,1,1], colorSpace='rgb', opacity=None,
    flipHoriz=False, flipVert=False,
    texRes=128.0, interpolate=True, depth=-2.0)
faceStim_neutral_neutral = parallel.ParallelPort(address='0x3FF8')
faceStim_happy_neutral = parallel.ParallelPort(address='0x3FF8')
faceStim_fearful_neutral = parallel.ParallelPort(address='0x3FF8')
fixation_cross_2 = visual.ShapeStim(
    win=win, name='fixation_cross_2', vertices='cross',units='pix',
    size=(35, 35),
    ori=0.0, pos=(0, 0), anchor='center',
    lineWidth=0.02,     colorSpace='rgb',  lineColor='white', fillColor='white',
    opacity=None, depth=-6.0, interpolate=True)

# --- Initialize components for Routine "Dot_Stimuli" ---
dotStim_congurent = parallel.ParallelPort(address='0x3FF8')
dotStim_incongurent = parallel.ParallelPort(address='0x3FF8')
key_reaction = keyboard.Keyboard()
dot1_left = visual.ShapeStim(
    win=win, name='dot1_left',units='pix',
    size=(25, 25), vertices='circle',
    ori=0.0, pos=(-687, 0), anchor='center',
    lineWidth=1.0,     colorSpace='rgb',  lineColor='white', fillColor='white',
    opacity=None, depth=-4.0, interpolate=True)
dot1_right = visual.ShapeStim(
    win=win, name='dot1_right',units='pix',
    size=(25, 25), vertices='circle',
    ori=0.0, pos=(687, 0), anchor='center',
    lineWidth=1.0,     colorSpace='rgb',  lineColor='white', fillColor='white',
    opacity=None, depth=-5.0, interpolate=True)
fixation_cross_3 = visual.ShapeStim(
    win=win, name='fixation_cross_3', vertices='cross',units='pix',
    size=(35, 35),
    ori=0.0, pos=(0, 0), anchor='center',
    lineWidth=0.02,     colorSpace='rgb',  lineColor='white', fillColor='white',
    opacity=None, depth=-6.0, interpolate=True)

# --- Initialize components for Routine "Rest" ---
reaction = parallel.ParallelPort(address='0x3FF8')
reaction_time_text = visual.TextStim(win=win, name='reaction_time_text',
    text='Reaction time:',
    font='Open Sans',
    pos=(0, 0), height=0.05, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=-2.0);
reaction_time_numbers = visual.TextStim(win=win, name='reaction_time_numbers',
    text='',
    font='Open Sans',
    pos=(0, -0.1), height=0.05, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=-3.0);

# --- Initialize components for Routine "dot_probe_ended_new_experiment_start" ---
dot_probe_ended_text = visual.TextStim(win=win, name='dot_probe_ended_text',
    text="Reaction time experiment finnished.\n\nPress 'L' or 'R' to continue.",
    font='Open Sans',
    pos=(0, 0), height=0.04, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=0.0);
Experiment_ended = parallel.ParallelPort(address='0x3FF8')
experiment_phase_2_key = keyboard.Keyboard()

# --- Initialize components for Routine "smooth_pursuit_info" ---
smooth_pursuit_info_text = visual.TextStim(win=win,
name='smooth_pursuit_info_text',
    text="Next experiement starts:\n\nIn the next task please follow the dot as best you
can.\n\nPress 'L' or 'R' when ready",
    font='Open Sans',
    pos=(0, 0), height=0.04, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=0.0);
```

```
smooth_pursuit_info_keyboard = keyboard.Keyboard()


# --- Initialize components for Routine "smooth_pursuit" ---
polygon_smooth = visual.ShapeStim(
    win=win, name='polygon_smooth',
    size=(0.05, 0.05), vertices='circle',
    ori=0.0, pos=(0, 0), anchor='center',
    lineWidth=1.0,     colorSpace='rgb',  lineColor='white', fillColor='white',
    opacity=None, depth=0.0, interpolate=True)


# --- Initialize components for Routine "saccade" ---
polygon_saccade = visual.ShapeStim(
    win=win, name='polygon_saccade',
    size=(0.05, 0.05), vertices='circle',
    ori=0.0, pos=(0, 0), anchor='center',
    lineWidth=1.0,     colorSpace='rgb',  lineColor='white', fillColor='white',
    opacity=None, depth=-1.0, interpolate=True)


# --- Initialize components for Routine "intro_waldo" ---
intro_waldo_picture = visual.ImageStim(
    win=win,
    name='intro_waldo_picture',
    image='AI_lab_exp/Intro.png', mask=None, anchor='center',
    ori=0.0, pos=(0, 0), size=(1, 1),
    color=[1,1,1], colorSpace='rgb', opacity=None,
    flipHoriz=False, flipVert=False,
    texRes=128.0, interpolate=True, depth=0.0)
intro_waldo_keyboard = keyboard.Keyboard()


# --- Initialize components for Routine "waldo" ---
image = visual.ImageStim(
    win=win,
    name='image',
    image='AI_lab_exp/wiw1.jpg', mask=None, anchor='center',
    ori=0.0, pos=(0, 0), size=(1, 1),
    color=[1,1,1], colorSpace='rgb', opacity=None,
    flipHoriz=False, flipVert=False,
    texRes=128.0, interpolate=True, depth=0.0)
image_2 = visual.ImageStim(
    win=win,
    name='image_2',
    image='AI_lab_exp/wiw2.png', mask=None, anchor='center',
    ori=0.0, pos=(0, 0), size=(1, 1),
    color=[1,1,1], colorSpace='rgb', opacity=None,
    flipHoriz=False, flipVert=False,
    texRes=128.0, interpolate=True, depth=-1.0)
image_3 = visual.ImageStim(
    win=win,
    name='image_3',
    image='AI_lab_exp/wiw3.jpg', mask=None, anchor='center',
    ori=0.0, pos=(0, 0), size=(1,1),
    color=[1,1,1], colorSpace='rgb', opacity=None,
    flipHoriz=False, flipVert=False,
    texRes=128.0, interpolate=True, depth=-2.0)
image_4 = visual.ImageStim(
    win=win,
    name='image_4',
    image='AI_lab_exp/wiw4.jpg', mask=None, anchor='center',
    ori=0.0, pos=(0, 0), size=(1, 1),
    color=[1,1,1], colorSpace='rgb', opacity=None,
    flipHoriz=False, flipVert=False,
    texRes=128.0, interpolate=True, depth=-3.0)
image_5 = visual.ImageStim(
    win=win,
    name='image_5',
    image='AI_lab_exp/wiw5.jpg', mask=None, anchor='center',
    ori=0.0, pos=(0, 0), size=(1, 1),
    color=[1,1,1], colorSpace='rgb', opacity=None,
    flipHoriz=False, flipVert=False,
    texRes=128.0, interpolate=True, depth=-4.0)
image_6 = visual.ImageStim(
    win=win,
    name='image_6',
    image='AI_lab_exp/wiw6.jpg', mask=None, anchor='center',
    ori=0.0, pos=(0, 0), size=(1, 1),
    color=[1,1,1], colorSpace='rgb', opacity=None,
    flipHoriz=False, flipVert=False,
    texRes=128.0, interpolate=True, depth=-5.0)
```

```
# --- Initialize components for Routine "intro_rand_pix" ---
text_3 = visual.TextStim(win=win, name='text_3',
    text="Look carefully at the following images.\n\nPress 'L' or 'R' when you are
ready.",
    font='Open Sans',
    pos=(0, 0), height=0.05, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=0.0);
key_resp_2 = keyboard.Keyboard()

# --- Initialize components for Routine "rand_pix" ---
image_7 = visual.ImageStim(
    win=win,
    name='image_7',
    image='AI_lab_exp/randpix0.jpg', mask=None, anchor='center',
    ori=0.0, pos=(0, 0), size=(1, 1),
    color=[1,1,1], colorSpace='rgb', opacity=None,
    flipHoriz=False, flipVert=False,
    texRes=128.0, interpolate=True, depth=0.0)
image_8 = visual.ImageStim(
    win=win,
    name='image_8',
    image='AI_lab_exp/randpix1.jpg', mask=None, anchor='center',
    ori=0.0, pos=(0, 0), size=(1, 1),
    color=[1,1,1], colorSpace='rgb', opacity=None,
    flipHoriz=False, flipVert=False,
    texRes=128.0, interpolate=True, depth=-1.0)
image_9 = visual.ImageStim(
    win=win,
    name='image_9',
    image='AI_lab_exp/randpix2.jpg', mask=None, anchor='center',
    ori=0.0, pos=(0, 0), size=(1, 1),
    color=[1,1,1], colorSpace='rgb', opacity=None,
    flipHoriz=False, flipVert=False,
    texRes=128.0, interpolate=True, depth=-2.0)
image_10 = visual.ImageStim(
    win=win,
    name='image_10',
    image='AI_lab_exp/randpix4.jpg', mask=None, anchor='center',
    ori=0.0, pos=(0, 0), size=(1, 1),
    color=[1,1,1], colorSpace='rgb', opacity=None,
    flipHoriz=False, flipVert=False,
    texRes=128.0, interpolate=True, depth=-3.0)

# --- Initialize components for Routine "End" ---
End_text = visual.TextStim(win=win, name='End_text',
    text='Experiment is over\nThank you ',
    font='Open Sans',
    pos=(0, 0), height=0.1, wrapWidth=None, ori=0.0,
    color='white', colorSpace='rgb', opacity=None,
    languageStyle='LTR',
    depth=0.0);

# Create some handy timers
globalClock = core.Clock()  # to track the time since experiment started
routineTimer = core.Clock()  # to track time remaining of each (possibly non-slip)
routine

# --- Prepare to start Routine "start_and_eyetracking_calibration" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
# keep track of which components have finished
start_and_eyetracking_calibrationComponents = []
for thisComponent in start_and_eyetracking_calibrationComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1
```

```
# --- Run Routine "start_and_eyetracking_calibration" ---
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in start_and_eyetracking_calibrationComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished

    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()
```

```
# --- Ending Routine "start_and_eyetracking_calibration" ---
for thisComponent in start_and_eyetracking_calibrationComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
# Run 'End Routine' code from Start_and_end_code
#Hides the mouse when the experiment is running
event.Mouse(visible=False)



# the Routine "start_and_eyetracking_calibration" was not non-slip safe, so reset the
non-slip timer
routineTimer.reset()

# --- Prepare to start Routine "vas_measure" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
vas_response.keys = []
vas_response.rt = []
_vas_response_allKeys = []
# keep track of which components have finished
vas_measureComponents = [vas_text, vas_response]
for thisComponent in vas_measureComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "vas_measure" ---
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # *vas_text* updates
    if vas_text.status == NOT_STARTED and tThisFlip >= 0.0-frameTolerance:
        # keep track of start time/frame for later
        vas_text.frameNStart = frameN  # exact frame index
        vas_text.tStart = t  # local t and not account for scr refresh
        vas_text.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(vas_text, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'vas_text.started')
        vas_text.setAutoDraw(True)

    # *vas_response* updates
    waitOnFlip = False
    if vas_response.status == NOT_STARTED and tThisFlip >= 0.0-frameTolerance:
        # keep track of start time/frame for later
        vas_response.frameNStart = frameN  # exact frame index
        vas_response.tStart = t  # local t and not account for scr refresh
        vas_response.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(vas_response, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'vas_response.started')
        vas_response.status = STARTED
        # keyboard checking is just starting
        waitOnFlip = True
        win.callOnFlip(vas_response.clock.reset)  # t=0 on next screen flip
        win.callOnFlip(vas_response.clearEvents, eventType='keyboard')  # clear
events on next screen flip
    if vas_response.status == STARTED and not waitOnFlip:
        theseKeys = vas_response.getKeys(keyList=['1','2','3','4','5','6','7','8','9'],
waitRelease=False)
        _vas_response_allKeys.extend(theseKeys)
        if len(_vas_response_allKeys):
            vas_response.keys = _vas_response_allKeys[-1].name  # just the last key
pressed
```

```
            vas_response.rt = _vas_response_allKeys[-1].rt
            # a response ends the routine
            continueRoutine = False

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in vas_measureComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished

    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()

# --- Ending Routine "vas_measure" ---
for thisComponent in vas_measureComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
```

```
# Run 'End Routine' code from vas_code
VAS = vas_response.keys
# check responses
if vas_response.keys in ['', [], None]:  # No response was made
    vas_response.keys = None
thisExp.addData('vas_response.keys',vas_response.keys)
if vas_response.keys != None:  # we had a response
    thisExp.addData('vas_response.rt', vas_response.rt)
thisExp.nextEntry()
# the Routine "vas_measure" was not non-slip safe, so reset the non-slip timer
routineTimer.reset()

# --- Prepare to start Routine "start_eeg_reminder" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
eeg_data_collection_on_key.keys = []
eeg_data_collection_on_key.rt = []
_eeg_data_collection_on_key_allKeys = []
# keep track of which components have finished
start_eeg_reminderComponents = [reminder_eeg_data_collection_text,
eeg_data_collection_on_key]
for thisComponent in start_eeg_reminderComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "start_eeg_reminder" ---
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # *reminder_eeg_data_collection_text* updates
    if reminder_eeg_data_collection_text.status == NOT_STARTED and tThisFlip >=
0.0-frameTolerance:
        # keep track of start time/frame for later
        reminder_eeg_data_collection_text.frameNStart = frameN  # exact frame index
        reminder_eeg_data_collection_text.tStart = t  # local t and not account for scr
refresh
        reminder_eeg_data_collection_text.tStartRefresh = tThisFlipGlobal  # on global
time
```

```
        win.timeOnFlip(reminder_eeg_data_collection_text, 'tStartRefresh')  # time at
next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'reminder_eeg_data_collection_text.started')
        reminder_eeg_data_collection_text.setAutoDraw(True)

    # *eeg_data_collection_on_key* updates
    waitOnFlip = False
    if eeg_data_collection_on_key.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
        # keep track of start time/frame for later
        eeg_data_collection_on_key.frameNStart = frameN  # exact frame index
        eeg_data_collection_on_key.tStart = t  # local t and not account for scr refresh
        eeg_data_collection_on_key.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(eeg_data_collection_on_key, 'tStartRefresh')  # time at next scr
refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'eeg_data_collection_on_key.started')
        eeg_data_collection_on_key.status = STARTED
        # keyboard checking is just starting
        waitOnFlip = True
        win.callOnFlip(eeg_data_collection_on_key.clock.reset)  # t=0 on next screen
flip
        win.callOnFlip(eeg_data_collection_on_key.clearEvents, eventType='keyboard')
# clear events on next screen flip
    if eeg_data_collection_on_key.status == STARTED and not waitOnFlip:
        theseKeys = eeg_data_collection_on_key.getKeys(keyList=['space'],
waitRelease=False)
        _eeg_data_collection_on_key_allKeys.extend(theseKeys)
        if len(_eeg_data_collection_on_key_allKeys):
            eeg_data_collection_on_key.keys = _eeg_data_collection_on_key_allKeys[-
1].name  # just the last key pressed
            eeg_data_collection_on_key.rt = _eeg_data_collection_on_key_allKeys[-1].rt
            # a response ends the routine
            continueRoutine = False

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in start_eeg_reminderComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished

    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
```

```
        win.flip()


# --- Ending Routine "start_eeg_reminder" ---
for thisComponent in start_eeg_reminderComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
# check responses
if eeg_data_collection_on_key.keys in ['', [], None]:  # No response was made
    eeg_data_collection_on_key.keys = None
thisExp.addData('eeg_data_collection_on_key.keys',eeg_data_collection_on_key.ke
ys)
if eeg_data_collection_on_key.keys != None:  # we had a response
    thisExp.addData('eeg_data_collection_on_key.rt', eeg_data_collection_on_key.rt)
thisExp.nextEntry()
# the Routine "start_eeg_reminder" was not non-slip safe, so reset the non-slip timer
routineTimer.reset()


# --- Prepare to start Routine "rest_state_OPEN_eye_info" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
rest_state_open_key.keys = []
rest_state_open_key.rt = []
_rest_state_open_key_allKeys = []
# keep track of which components have finished
```

```
rest_state_OPEN_eye_infoComponents = [rest_state_open_text,
rest_state_open_key]
for thisComponent in rest_state_OPEN_eye_infoComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "rest_state_OPEN_eye_info" ---
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # *rest_state_open_text* updates
    if rest_state_open_text.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
```

```
            # keep track of start time/frame for later
            rest_state_open_text.frameNStart = frameN  # exact frame index
            rest_state_open_text.tStart = t  # local t and not account for scr refresh
            rest_state_open_text.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(rest_state_open_text, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'rest_state_open_text.started')
            rest_state_open_text.setAutoDraw(True)

        # *rest_state_open_key* updates
        waitOnFlip = False
        if rest_state_open_key.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
            # keep track of start time/frame for later
            rest_state_open_key.frameNStart = frameN  # exact frame index
            rest_state_open_key.tStart = t  # local t and not account for scr refresh
            rest_state_open_key.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(rest_state_open_key, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'rest_state_open_key.started')
            rest_state_open_key.status = STARTED
            # keyboard checking is just starting
            waitOnFlip = True
            win.callOnFlip(rest_state_open_key.clock.reset)  # t=0 on next screen flip
            win.callOnFlip(rest_state_open_key.clearEvents, eventType='keyboard')  # clear
events on next screen flip
```

```
    if rest_state_open_key.status == STARTED and not waitOnFlip:
        theseKeys = rest_state_open_key.getKeys(keyList=['1','2'], waitRelease=False)
        _rest_state_open_key_allKeys.extend(theseKeys)
        if len(_rest_state_open_key_allKeys):
            rest_state_open_key.keys = _rest_state_open_key_allKeys[-1].name  # just
the last key pressed
            rest_state_open_key.rt = _rest_state_open_key_allKeys[-1].rt
            # a response ends the routine
            continueRoutine = False

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in rest_state_OPEN_eye_infoComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished

    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
```

```
        win.flip()


# --- Ending Routine "rest_state_OPEN_eye_info" ---
for thisComponent in rest_state_OPEN_eye_infoComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
# check responses
if rest_state_open_key.keys in ['', [], None]:  # No response was made
    rest_state_open_key.keys = None
thisExp.addData('rest_state_open_key.keys',rest_state_open_key.keys)
if rest_state_open_key.keys != None:  # we had a response
    thisExp.addData('rest_state_open_key.rt', rest_state_open_key.rt)
thisExp.nextEntry()
# the Routine "rest_state_OPEN_eye_info" was not non-slip safe, so reset the non-
slip timer
routineTimer.reset()


# --- Prepare to start Routine "rest_state_OPEN_eye_start_counter" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
# keep track of which components have finished
rest_state_OPEN_eye_start_counterComponents = []
for thisComponent in rest_state_OPEN_eye_start_counterComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "rest_state_OPEN_eye_start_counter" ---
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in rest_state_OPEN_eye_start_counterComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished

    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()

# --- Ending Routine "rest_state_OPEN_eye_start_counter" ---
for thisComponent in rest_state_OPEN_eye_start_counterComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
# the Routine "rest_state_OPEN_eye_start_counter" was not non-slip safe, so reset
the non-slip timer
routineTimer.reset()

# --- Prepare to start Routine "rest_state_OPEN_eye_TASK" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
# keep track of which components have finished
rest_state_OPEN_eye_TASKComponents = []
for thisComponent in rest_state_OPEN_eye_TASKComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "rest_state_OPEN_eye_TASK" ---
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in rest_state_OPEN_eye_TASKComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished

    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()

# --- Ending Routine "rest_state_OPEN_eye_TASK" ---
for thisComponent in rest_state_OPEN_eye_TASKComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
# the Routine "rest_state_OPEN_eye_TASK" was not non-slip safe, so reset the non-
slip timer
routineTimer.reset()

# --- Prepare to start Routine "rest_state_CLOSED_eye_info" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
rest_state_closed_key.keys = []
rest_state_closed_key.rt = []
_rest_state_closed_key_allKeys = []
```

```
# keep track of which components have finished
rest_state_CLOSED_eye_infoComponents = [rest_state_closed_text,
rest_state_closed_key]
for thisComponent in rest_state_CLOSED_eye_infoComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "rest_state_CLOSED_eye_info" ---
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # *rest_state_closed_text* updates
    if rest_state_closed_text.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
        # keep track of start time/frame for later
        rest_state_closed_text.frameNStart = frameN  # exact frame index
        rest_state_closed_text.tStart = t  # local t and not account for scr refresh
        rest_state_closed_text.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(rest_state_closed_text, 'tStartRefresh')  # time at next scr
refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'rest_state_closed_text.started')
        rest_state_closed_text.setAutoDraw(True)

    # *rest_state_closed_key* updates
    waitOnFlip = False
    if rest_state_closed_key.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
        # keep track of start time/frame for later
        rest_state_closed_key.frameNStart = frameN  # exact frame index
        rest_state_closed_key.tStart = t  # local t and not account for scr refresh
        rest_state_closed_key.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(rest_state_closed_key, 'tStartRefresh')  # time at next scr
refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'rest_state_closed_key.started')
        rest_state_closed_key.status = STARTED
        # keyboard checking is just starting
        waitOnFlip = True
        win.callOnFlip(rest_state_closed_key.clock.reset)  # t=0 on next screen flip
        win.callOnFlip(rest_state_closed_key.clearEvents, eventType='keyboard')  #
clear events on next screen flip
    if rest_state_closed_key.status == STARTED and not waitOnFlip:
        theseKeys = rest_state_closed_key.getKeys(keyList=['1','2'],
waitRelease=False)
        _rest_state_closed_key_allKeys.extend(theseKeys)
        if len(_rest_state_closed_key_allKeys):
            rest_state_closed_key.keys = _rest_state_closed_key_allKeys[-1].name  #
just the last key pressed
            rest_state_closed_key.rt = _rest_state_closed_key_allKeys[-1].rt
            # a response ends the routine
            continueRoutine = False

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in rest_state_CLOSED_eye_infoComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished
    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()

# --- Ending Routine "rest_state_CLOSED_eye_info" ---
for thisComponent in rest_state_CLOSED_eye_infoComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
# check responses
if rest_state_closed_key.keys in ['', [], None]:  # No response was made
    rest_state_closed_key.keys = None
thisExp.addData('rest_state_closed_key.keys',rest_state_closed_key.keys)
if rest_state_closed_key.keys != None:  # we had a response
    thisExp.addData('rest_state_closed_key.rt', rest_state_closed_key.rt)
thisExp.nextEntry()
# the Routine "rest_state_CLOSED_eye_info" was not non-slip safe, so reset the
non-slip timer
routineTimer.reset()

# --- Prepare to start Routine "rest_state_CLOSED_eye_start_counter" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
# keep track of which components have finished
rest_state_CLOSED_eye_start_counterComponents = []
```

```
for thisComponent in rest_state_CLOSED_eye_start_counterComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "rest_state_CLOSED_eye_start_counter" ---
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in rest_state_CLOSED_eye_start_counterComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished

    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()

# --- Ending Routine "rest_state_CLOSED_eye_start_counter" ---
for thisComponent in rest_state_CLOSED_eye_start_counterComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
# the Routine "rest_state_CLOSED_eye_start_counter" was not non-slip safe, so
reset the non-slip timer
routineTimer.reset()

# --- Prepare to start Routine "rest_state_CLOSED_eye_TASK" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
# keep track of which components have finished
```

```
rest_state_CLOSED_eye_TASKComponents = []
for thisComponent in rest_state_CLOSED_eye_TASKComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "rest_state_CLOSED_eye_TASK" ---
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()
    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in rest_state_CLOSED_eye_TASKComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished

    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()

# --- Ending Routine "rest_state_CLOSED_eye_TASK" ---
for thisComponent in rest_state_CLOSED_eye_TASKComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
# the Routine "rest_state_CLOSED_eye_TASK" was not non-slip safe, so reset the
non-slip timer
routineTimer.reset()

# --- Prepare to start Routine "dot_probe_task_info" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
```

```
task_info_response.keys = []
task_info_response.rt = []
_task_info_response_allKeys = []
# keep track of which components have finished
dot_probe_task_infoComponents = [task_info_text, task_info_response]
for thisComponent in dot_probe_task_infoComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "dot_probe_task_info" ---
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame
    # *task_info_text* updates
    if task_info_text.status == NOT_STARTED and tThisFlip >= 0.0-frameTolerance:
        # keep track of start time/frame for later
        task_info_text.frameNStart = frameN  # exact frame index
        task_info_text.tStart = t  # local t and not account for scr refresh
        task_info_text.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(task_info_text, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'task_info_text.started')
        task_info_text.setAutoDraw(True)

    # *task_info_response* updates
    waitOnFlip = False
    if task_info_response.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
        # keep track of start time/frame for later
        task_info_response.frameNStart = frameN  # exact frame index
        task_info_response.tStart = t  # local t and not account for scr refresh
        task_info_response.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(task_info_response, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'task_info_response.started')
        task_info_response.status = STARTED
        # keyboard checking is just starting
        waitOnFlip = True
        win.callOnFlip(task_info_response.clock.reset)  # t=0 on next screen flip
        win.callOnFlip(task_info_response.clearEvents, eventType='keyboard')  # clear
events on next screen flip
    if task_info_response.status == STARTED and not waitOnFlip:
        theseKeys = task_info_response.getKeys(keyList=['1','2'], waitRelease=False)
        _task_info_response_allKeys.extend(theseKeys)
        if len(_task_info_response_allKeys):
            task_info_response.keys = _task_info_response_allKeys[-1].name  # just the
last key pressed
            task_info_response.rt = _task_info_response_allKeys[-1].rt
            # a response ends the routine
            continueRoutine = False

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in dot_probe_task_infoComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished
    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()

# --- Ending Routine "dot_probe_task_info" ---
for thisComponent in dot_probe_task_infoComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
# check responses
if task_info_response.keys in ['', [], None]:  # No response was made
    task_info_response.keys = None
thisExp.addData('task_info_response.keys',task_info_response.keys)
if task_info_response.keys != None:  # we had a response
    thisExp.addData('task_info_response.rt', task_info_response.rt)
thisExp.nextEntry()
# the Routine "dot_probe_task_info" was not non-slip safe, so reset the non-slip timer
routineTimer.reset()

# --- Prepare to start Routine "dot_probe_start_counter" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
# Run 'Begin Routine' code from start_eye_track_recording_and_clock_code
# get a reference to the currently active EyeLink connection
el_tracker = pylink.getEYELINK()
```

```
# put the tracker in the offline mode first
el_tracker.setOfflineMode()

# clear the host screen before we draw the backdrop
el_tracker.sendCommand('clear_screen 0')

trial_index = 0


# put tracker in idle/offline mode before recording
el_tracker.setOfflineMode()

# Start recording
# arguments: sample_to_file, events_to_file, sample_over_link,
# event_over_link (1-yes, 0-no)
try:
   el_tracker.startRecording(1, 1, 1, 1)
except RuntimeError as error:
   print("ERROR:", error)
   abort_trial()
# keep track of which components have finished
dot_probe_start_counterComponents = [Start_experiment, dot_probe_text_counter,
dot_probe_start_text]
for thisComponent in dot_probe_start_counterComponents:
   thisComponent.tStart = None
   thisComponent.tStop = None
```

```
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "dot_probe_start_counter" ---
while continueRoutine and routineTimer.getTime() < 15.0:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame
    # *Start_experiment* updates
    if Start_experiment.status == NOT_STARTED and tThisFlip >= 15-
frameTolerance:
        # keep track of start time/frame for later
        Start_experiment.frameNStart = frameN  # exact frame index
        Start_experiment.tStart = t  # local t and not account for scr refresh
        Start_experiment.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(Start_experiment, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'Start_experiment.started')
        Start_experiment.status = STARTED
        win.callOnFlip(Start_experiment.setData, int(1))
    if Start_experiment.status == STARTED:
        if frameN >= (Start_experiment.frameNStart + 2.0):
            # keep track of stop time/frame for later
            Start_experiment.tStop = t  # not accounting for scr refresh
            Start_experiment.frameNStop = frameN  # exact frame index
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'Start_experiment.stopped')
            Start_experiment.status = FINISHED
            win.callOnFlip(Start_experiment.setData, int(0))

    # *dot_probe_text_counter* updates
    if dot_probe_text_counter.status == NOT_STARTED and tThisFlip >= 0-
frameTolerance:
        # keep track of start time/frame for later
        dot_probe_text_counter.frameNStart = frameN  # exact frame index
        dot_probe_text_counter.tStart = t  # local t and not account for scr refresh
        dot_probe_text_counter.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(dot_probe_text_counter, 'tStartRefresh')  # time at next scr
refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'dot_probe_text_counter.started')
        dot_probe_text_counter.setAutoDraw(True)
    if dot_probe_text_counter.status == STARTED:
        # is it time to stop? (based on global clock, using actual start)
        if tThisFlipGlobal > dot_probe_text_counter.tStartRefresh + 15-frameTolerance:
            # keep track of stop time/frame for later
            dot_probe_text_counter.tStop = t  # not accounting for scr refresh
            dot_probe_text_counter.frameNStop = frameN  # exact frame index
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'dot_probe_text_counter.stopped')
            dot_probe_text_counter.setAutoDraw(False)
    if dot_probe_text_counter.status == STARTED:  # only update if drawing
        dot_probe_text_counter.setText(round(15.0 - t, ndigits = 0)


, log=False)


    # *dot_probe_start_text* updates
    if dot_probe_start_text.status == NOT_STARTED and tThisFlip >= 0-
frameTolerance:
        # keep track of start time/frame for later
        dot_probe_start_text.frameNStart = frameN  # exact frame index
        dot_probe_start_text.tStart = t  # local t and not account for scr refresh
        dot_probe_start_text.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(dot_probe_start_text, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'dot_probe_start_text.started')
        dot_probe_start_text.setAutoDraw(True)
    if dot_probe_start_text.status == STARTED:
```

```
    # is it time to stop? (based on global clock, using actual start)
    if tThisFlipGlobal > dot_probe_start_text.tStartRefresh + 15-frameTolerance:
        # keep track of stop time/frame for later
        dot_probe_start_text.tStop = t  # not accounting for scr refresh
        dot_probe_start_text.frameNStop = frameN  # exact frame index
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'dot_probe_start_text.stopped')
        dot_probe_start_text.setAutoDraw(False)

# check for quit (typically the Esc key)
if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
    core.quit()

# check if all components have finished
if not continueRoutine:  # a component has requested a forced-end of Routine
    routineForceEnded = True
    break
continueRoutine = False  # will revert to True if at least one component still running
for thisComponent in dot_probe_start_counterComponents:
    if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
        continueRoutine = True
        break  # at least one component has not yet finished

# refresh the screen
if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
    win.flip()
```

```
# --- Ending Routine "dot_probe_start_counter" ---
for thisComponent in dot_probe_start_counterComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
if Start_experiment.status == STARTED:
    win.callOnFlip(Start_experiment.setData, int(0))
# Run 'End Routine' code from start_eye_track_recording_and_clock_code
#Starting timer. It starts when the image routines begins.
expClock = core.Clock()

#send message to Dataviewer the routine is starting
el_tracker.sendMessage('beginExperiment')

# using non-slip timing so subtract the expected duration of this Routine (unless
ended on request)
if routineForceEnded:
    routineTimer.reset()
else:
    routineTimer.addTime(-15.000000)

# set up handler to look after randomisation of conditions etc
thisTrial = data.TrialHandler(nReps=2.0, method='random',
    extraInfo=expInfo, originPath=-1,
    trialList=data.importConditions('picture_setup_dot_probe_eeg_eye_tracker.xlsx'),
    seed=None, name='thisTrial')
```

```
thisExp.addLoop(thisTrial)  # add the loop to the experiment
thisThisTrial = thisTrial.trialList[0]  # so we can initialise stimuli with some values
# abbreviate parameter names if possible (e.g. rgb = thisThisTrial.rgb)
if thisThisTrial != None:
    for paramName in thisThisTrial:
        exec('{} = thisThisTrial[paramName]'.format(paramName))

for thisThisTrial in thisTrial:
    currentLoop = thisTrial
    # abbreviate parameter names if possible (e.g. rgb = thisThisTrial.rgb)
    if thisThisTrial != None:
        for paramName in thisThisTrial:
            exec('{} = thisThisTrial[paramName]'.format(paramName))

    # --- Prepare to start Routine "Fixation_Cross" ---
    continueRoutine = True
    routineForceEnded = False
    # update component parameters for each repeat
    # Run 'Begin Routine' code from fixation_cross_code
    #Sending trail info to the eye tracker
    el_tracker.sendMessage('TRIALID %d' % trial_index)

    trial_index += 1

    #el_tracker.sendMessage(trial_initial_info)
    el_tracker.sendMessage('Fixation_Cross_Start')
    el_tracker.sendMessage('!V DRAWLINE 255 255 255 960 505 960 575')
    el_tracker.sendMessage('!V DRAWLINE 255 255 255 925 540 995 540')


    # keep track of which components have finished
    Fixation_CrossComponents = [fixation_cross, fixation]
    for thisComponent in Fixation_CrossComponents:
        thisComponent.tStart = None
        thisComponent.tStop = None
        thisComponent.tStartRefresh = None
        thisComponent.tStopRefresh = None
        if hasattr(thisComponent, 'status'):
            thisComponent.status = NOT_STARTED
    # reset timers
    t = 0
    _timeToFirstFrame = win.getFutureFlipTime(clock="now")
    frameN = -1

    # --- Run Routine "Fixation_Cross" ---
    while continueRoutine:
        # get current time
        t = routineTimer.getTime()
        tThisFlip = win.getFutureFlipTime(clock=routineTimer)
        tThisFlipGlobal = win.getFutureFlipTime(clock=None)
        frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
```

```
        # update/draw components on each frame
        # Run 'Each Frame' code from fixation_cross_code
        if t > Time_Jitter:
            break


        # *fixation_cross* updates
        if fixation_cross.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
            # keep track of start time/frame for later
            fixation_cross.frameNStart = frameN  # exact frame index
            fixation_cross.tStart = t  # local t and not account for scr refresh
            fixation_cross.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(fixation_cross, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'fixation_cross.started')
            fixation_cross.setAutoDraw(True)
        # *fixation* updates
        if fixation.status == NOT_STARTED and tThisFlip >= 0.0-frameTolerance:
            # keep track of start time/frame for later
            fixation.frameNStart = frameN  # exact frame index
            fixation.tStart = t  # local t and not account for scr refresh
            fixation.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(fixation, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'fixation.started')
            fixation.status = STARTED
            win.callOnFlip(fixation.setData, int(2))
        if fixation.status == STARTED:
            if frameN >= (fixation.frameNStart + 2.0):
                # keep track of stop time/frame for later
                fixation.tStop = t  # not accounting for scr refresh
                fixation.frameNStop = frameN  # exact frame index
                # add timestamp to datafile
                thisExp.timestampOnFlip(win, 'fixation.stopped')
                fixation.status = FINISHED
                win.callOnFlip(fixation.setData, int(0))

        # check for quit (typically the Esc key)
        if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
            core.quit()

        # check if all components have finished
        if not continueRoutine:  # a component has requested a forced-end of Routine
            routineForceEnded = True
            break
        continueRoutine = False  # will revert to True if at least one component still
running
        for thisComponent in Fixation_CrossComponents:
            if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
                continueRoutine = True
                break  # at least one component has not yet finished
        # refresh the screen
        if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
            win.flip()

    # --- Ending Routine "Fixation_Cross" ---
    for thisComponent in Fixation_CrossComponents:
        if hasattr(thisComponent, "setAutoDraw"):
            thisComponent.setAutoDraw(False)
    # Run 'End Routine' code from fixation_cross_code
    el_tracker.sendMessage('Fixation_Cross_Stop')
    if fixation.status == STARTED:
        win.callOnFlip(fixation.setData, int(0))
    # the Routine "Fixation_Cross" was not non-slip safe, so reset the non-slip timer
    routineTimer.reset()

    # --- Prepare to start Routine "Faces_Stimuli" ---
    continueRoutine = True
    routineForceEnded = False
    # update component parameters for each repeat
    # Run 'Begin Routine' code from code_faces_stimuli
    #To find ratio from psychopy take use this formula:(scn_width/2)- (psychopy_width
* scn_height)
    left_image_center_x_axis = int((scn_width/2)-(687))
    right_image_center_x_axis = int((scn_width/2)+(687))
    image_center_y_axis = int(scn_height/2.0)
    image_width = int(362)
    image_height = int(506)


    #'!V IMGLOAD CENTER %s %d %d %d %d' % (bg_image, int(scn_width/2.0),
int(scn_height/2.0), int(scn_width), int(scn_height))
    #!V IMGLOAD CENTER <relative_image_path> <x_position> <y_position> [width]
[height]
    image_face_left = "../../" + Face_Stimuli_Left
    image_face_right = "../../" + Face_Stimuli_Right


    el_tracker.sendMessage('!V IMGLOAD CENTER %s %d %d %d %d' %
(image_face_left, left_image_center_x_axis, image_center_y_axis, image_width,
image_height))
    el_tracker.sendMessage('!V IMGLOAD CENTER %s %d %d %d %d' %
(image_face_right, right_image_center_x_axis, image_center_y_axis, image_width,
image_height))


    left_image_left_border = left_image_center_x_axis - image_width/2 #left left
    left_image_right_border= left_image_center_x_axis + image_width/2 #left right
    right_image_left_border= right_image_center_x_axis - image_width/2 #right left
    right_image_right_border= right_image_center_x_axis + image_width/2 #right right
    top = image_center_y_axis + image_height/2  #top
    bottom= image_center_y_axis - image_height/2  #bottom


    # send interest area messages to record in the EDF data file
    # here we draw a rectangular IA, for illustration purposes
    # format: !V IAREA RECTANGLE <id> <left> <top> <right> <bottom> [label]
    # for all supported interest area commands, see the Data Viewer Manual,
    # "Protocol for EyeLink Data to Viewer Integration"
    ia_image_left = (1, left_image_left_border, top, left_image_right_border, bottom , 'square')
    ia_image_right = (2, right_image_left_border, top, right_image_right_border, bottom , 'square')


    el_tracker.sendMessage('!V IAREA RECTANGLE %d %d %d %d %d %s' % ia_image_left)
    el_tracker.sendMessage('!V IAREA RECTANGLE %d %d %d %d %d %s' % ia_image_right)



    el_tracker.sendMessage('Faces_Stimuli_Start')




    image_left.setImage(Face_Stimuli_Left)
    image_right.setImage(Face_Stimuli_Right)
    # keep track of which components have finished
    Faces_StimuliComponents = [image_left, image_right, faceStim_neutral_neutral, faceStim_happy_neutral, faceStim_fearful_neutral, fixation_cross_2]
    for thisComponent in Faces_StimuliComponents:
        thisComponent.tStart = None
        thisComponent.tStop = None
        thisComponent.tStartRefresh = None
        thisComponent.tStopRefresh = None
        if hasattr(thisComponent, 'status'):
```

```
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "Faces_Stimuli" ---
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame
    # Run 'Each Frame' code from code_faces_stimuli
    if t > 1.2:
        break

    # *image_left* updates
    if image_left.status == NOT_STARTED and tThisFlip >= 0-frameTolerance:
        # keep track of start time/frame for later
        image_left.frameNStart = frameN  # exact frame index
        image_left.tStart = t  # local t and not account for scr refresh
        image_left.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(image_left, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
```

```
            thisExp.timestampOnFlip(win, 'image_left.started')
            image_left.setAutoDraw(True)

        # *image_right* updates
        if image_right.status == NOT_STARTED and tThisFlip >= 0-frameTolerance:
            # keep track of start time/frame for later
            image_right.frameNStart = frameN  # exact frame index
            image_right.tStart = t  # local t and not account for scr refresh
            image_right.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(image_right, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'image_right.started')
            image_right.setAutoDraw(True)
        # *faceStim_neutral_neutral* updates
        if faceStim_neutral_neutral.status == NOT_STARTED and Face_Pairs ==
'neutral/neutral':
            # keep track of start time/frame for later
            faceStim_neutral_neutral.frameNStart = frameN  # exact frame index
            faceStim_neutral_neutral.tStart = t  # local t and not account for scr refresh
            faceStim_neutral_neutral.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(faceStim_neutral_neutral, 'tStartRefresh')  # time at next scr
refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'faceStim_neutral_neutral.started')
            faceStim_neutral_neutral.status = STARTED
            win.callOnFlip(faceStim_neutral_neutral.setData, int(4))
        if faceStim_neutral_neutral.status == STARTED:
            # is it time to stop? (based on global clock, using actual start)
            if tThisFlipGlobal > faceStim_neutral_neutral.tStartRefresh + 2.0-
frameTolerance:
                # keep track of stop time/frame for later
                faceStim_neutral_neutral.tStop = t  # not accounting for scr refresh
                faceStim_neutral_neutral.frameNStop = frameN  # exact frame index
                # add timestamp to datafile
                thisExp.timestampOnFlip(win, 'faceStim_neutral_neutral.stopped')
                faceStim_neutral_neutral.status = FINISHED
                win.callOnFlip(faceStim_neutral_neutral.setData, int(0))
        # *faceStim_happy_neutral* updates
        if faceStim_happy_neutral.status == NOT_STARTED and Face_Pairs ==
'happy/neutral':
            # keep track of start time/frame for later
            faceStim_happy_neutral.frameNStart = frameN  # exact frame index
            faceStim_happy_neutral.tStart = t  # local t and not account for scr refresh
            faceStim_happy_neutral.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(faceStim_happy_neutral, 'tStartRefresh')  # time at next scr
refresh
            faceStim_happy_neutral.status = STARTED
            faceStim_happy_neutral.setData(int(8))
        if faceStim_happy_neutral.status == STARTED:
            if frameN >= (faceStim_happy_neutral.frameNStart + 2.0):
                # keep track of stop time/frame for later
                faceStim_happy_neutral.tStop = t  # not accounting for scr refresh
                faceStim_happy_neutral.frameNStop = frameN  # exact frame index
            faceStim_happy_neutral.status = FINISHED
            faceStim_happy_neutral.setData(int(0))
        # *faceStim_fearful_neutral* updates
        if faceStim_fearful_neutral.status == NOT_STARTED and Face_Pairs == 
'fearful/neutral':
            # keep track of start time/frame for later
            faceStim_fearful_neutral.frameNStart = frameN  # exact frame index
            faceStim_fearful_neutral.tStart = t  # local t and not account for scr refresh
            faceStim_fearful_neutral.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(faceStim_fearful_neutral, 'tStartRefresh')  # time at next scr 
refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'faceStim_fearful_neutral.started')
            faceStim_fearful_neutral.status = STARTED
            win.callOnFlip(faceStim_fearful_neutral.setData, int(16))
        if faceStim_fearful_neutral.status == STARTED:
            if frameN >= (faceStim_fearful_neutral.frameNStart + 2.0):
                # keep track of stop time/frame for later
                faceStim_fearful_neutral.tStop = t  # not accounting for scr refresh
                faceStim_fearful_neutral.frameNStop = frameN  # exact frame index
                # add timestamp to datafile
                thisExp.timestampOnFlip(win, 'faceStim_fearful_neutral.stopped')
                faceStim_fearful_neutral.status = FINISHED
                win.callOnFlip(faceStim_fearful_neutral.setData, int(0))


        # *fixation_cross_2* updates
        if fixation_cross_2.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
            # keep track of start time/frame for later
            fixation_cross_2.frameNStart = frameN  # exact frame index
            fixation_cross_2.tStart = t  # local t and not account for scr refresh
            fixation_cross_2.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(fixation_cross_2, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'fixation_cross_2.started')
            fixation_cross_2.setAutoDraw(True)

        # check for quit (typically the Esc key)
        if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
            core.quit()

        # check if all components have finished
        if not continueRoutine:  # a component has requested a forced-end of Routine
            routineForceEnded = True
            break
        continueRoutine = False  # will revert to True if at least one component still
running
        for thisComponent in Faces_StimuliComponents:
            if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
                continueRoutine = True
                break  # at least one component has not yet finished
```

```
    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()

# --- Ending Routine "Faces_Stimuli" ---
for thisComponent in Faces_StimuliComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
# Run 'End Routine' code from code_faces_stimuli
el_tracker.sendMessage('Faces_Stimuli_Stop')

# Send a message to clear the Data Viewer screen
bgcolor_RGB = (0, 0, 0)
el_tracker.sendMessage('!V CLEAR %d %d %d' % bgcolor_RGB)



if faceStim_neutral_neutral.status == STARTED:
    win.callOnFlip(faceStim_neutral_neutral.setData, int(0))
if faceStim_happy_neutral.status == STARTED:
    faceStim_happy_neutral.setData(int(0))
if faceStim_fearful_neutral.status == STARTED:
    win.callOnFlip(faceStim_fearful_neutral.setData, int(0))
# the Routine "Faces_Stimuli" was not non-slip safe, so reset the non-slip timer
routineTimer.reset()

# --- Prepare to start Routine "Dot_Stimuli" ---
```

```
    continueRoutine = True
    routineForceEnded = False
    # update component parameters for each repeat
    # Run 'Begin Routine' code from code_dot_stimuli
    el_tracker.sendMessage('Dot_Stimuli_Start')



    if Dot_Location == 'left':
        show_dot_left = True

        #Drawing left dot to edf file
        el_tracker.sendMessage('!V DRAWLINE 255 255 255 %d %d %d %d' %
(left_image_center_x_axis, (image_center_y_axis - 20), left_image_center_x_axis,
(image_center_y_axis + 20)))
        el_tracker.sendMessage('!V DRAWLINE 255 255 255 %d %d %d %d' %
((left_image_center_x_axis - 20), image_center_y_axis, (left_image_center_x_axis +
20), image_center_y_axis))



    if Dot_Location == 'right':
        show_dot_right = True

        #Drawing right dot to edf
        el_tracker.sendMessage('!V DRAWLINE 255 255 255 %d %d %d %d' %
(right_image_center_x_axis, (image_center_y_axis - 20), right_image_center_x_axis,
(image_center_y_axis + 20)))
```

```
        el_tracker.sendMessage('!V DRAWLINE 255 255 255 %d %d %d %d' %
((right_image_center_x_axis - 20), image_center_y_axis,
(right_image_center_x_axis + 20), image_center_y_axis))


    key_reaction.keys = []
    key_reaction.rt = []
    _key_reaction_allKeys = []
    # keep track of which components have finished
    Dot_StimuliComponents = [dotStim_congurent, dotStim_incongurent,
key_reaction, dot1_left, dot1_right, fixation_cross_3]
    for thisComponent in Dot_StimuliComponents:
        thisComponent.tStart = None
        thisComponent.tStop = None
        thisComponent.tStartRefresh = None
        thisComponent.tStopRefresh = None
        if hasattr(thisComponent, 'status'):
            thisComponent.status = NOT_STARTED
    # reset timers
    t = 0
    _timeToFirstFrame = win.getFutureFlipTime(clock="now")
    frameN = -1


    # --- Run Routine "Dot_Stimuli" ---
    while continueRoutine:
        # get current time
        t = routineTimer.getTime()
        tThisFlip = win.getFutureFlipTime(clock=routineTimer)
```

```
tThisFlipGlobal = win.getFutureFlipTime(clock=None)
frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
# update/draw components on each frame
# *dotStim_congurent* updates
if dotStim_congurent.status == NOT_STARTED and Type == 'Congurent':
    # keep track of start time/frame for later
    dotStim_congurent.frameNStart = frameN  # exact frame index
    dotStim_congurent.tStart = t  # local t and not account for scr refresh
    dotStim_congurent.tStartRefresh = tThisFlipGlobal  # on global time
    win.timeOnFlip(dotStim_congurent, 'tStartRefresh')  # time at next scr refresh
    # add timestamp to datafile
    thisExp.timestampOnFlip(win, 'dotStim_congurent.started')
    dotStim_congurent.status = STARTED
    win.callOnFlip(dotStim_congurent.setData, int(32))
if dotStim_congurent.status == STARTED:
    if frameN >= (dotStim_congurent.frameNStart + 2.0):
        # keep track of stop time/frame for later
        dotStim_congurent.tStop = t  # not accounting for scr refresh
        dotStim_congurent.frameNStop = frameN  # exact frame index
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'dotStim_congurent.stopped')
        dotStim_congurent.status = FINISHED
        win.callOnFlip(dotStim_congurent.setData, int(0))
# *dotStim_incongurent* updates
if dotStim_incongurent.status == NOT_STARTED and Type == 'Incongurent':
    # keep track of start time/frame for later
```

```
            dotStim_incongurent.frameNStart = frameN  # exact frame index
            dotStim_incongurent.tStart = t  # local t and not account for scr refresh
            dotStim_incongurent.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(dotStim_incongurent, 'tStartRefresh')  # time at next scr
refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'dotStim_incongurent.started')
            dotStim_incongurent.status = STARTED
            win.callOnFlip(dotStim_incongurent.setData, int(64))
        if dotStim_incongurent.status == STARTED:
            if frameN >= (dotStim_incongurent.frameNStart + 2.0):
                # keep track of stop time/frame for later
                dotStim_incongurent.tStop = t  # not accounting for scr refresh
                dotStim_incongurent.frameNStop = frameN  # exact frame index
                # add timestamp to datafile
                thisExp.timestampOnFlip(win, 'dotStim_incongurent.stopped')
                dotStim_incongurent.status = FINISHED
                win.callOnFlip(dotStim_incongurent.setData, int(0))

        # *key_reaction* updates
        waitOnFlip = False
        if key_reaction.status == NOT_STARTED and tThisFlip >= 0-frameTolerance:
            # keep track of start time/frame for later
            key_reaction.frameNStart = frameN  # exact frame index
            key_reaction.tStart = t  # local t and not account for scr refresh
            key_reaction.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(key_reaction, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'key_reaction.started')
            key_reaction.status = STARTED
            # keyboard checking is just starting
            waitOnFlip = True
            win.callOnFlip(key_reaction.clock.reset)  # t=0 on next screen flip
            win.callOnFlip(key_reaction.clearEvents, eventType='keyboard')  # clear
events on next screen flip
        if key_reaction.status == STARTED and not waitOnFlip:
            theseKeys = key_reaction.getKeys(keyList=['1','2'], waitRelease=False)
            _key_reaction_allKeys.extend(theseKeys)
            if len(_key_reaction_allKeys):
                key_reaction.keys = _key_reaction_allKeys[0].name  # just the first key
pressed
                key_reaction.rt = _key_reaction_allKeys[0].rt
                # a response ends the routine
                continueRoutine = False

        # *dot1_left* updates
        if dot1_left.status == NOT_STARTED and show_dot_left == True:
            # keep track of start time/frame for later
            dot1_left.frameNStart = frameN  # exact frame index
            dot1_left.tStart = t  # local t and not account for scr refresh
            dot1_left.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(dot1_left, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'dot1_left.started')
            dot1_left.setAutoDraw(True)

        # *dot1_right* updates
        if dot1_right.status == NOT_STARTED and show_dot_right == True:
            # keep track of start time/frame for later
            dot1_right.frameNStart = frameN  # exact frame index
            dot1_right.tStart = t  # local t and not account for scr refresh
            dot1_right.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(dot1_right, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'dot1_right.started')
            dot1_right.setAutoDraw(True)

        # *fixation_cross_3* updates
        if fixation_cross_3.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
            # keep track of start time/frame for later
            fixation_cross_3.frameNStart = frameN  # exact frame index
            fixation_cross_3.tStart = t  # local t and not account for scr refresh
            fixation_cross_3.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(fixation_cross_3, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'fixation_cross_3.started')
            fixation_cross_3.setAutoDraw(True)
        # check for quit (typically the Esc key)
        if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
            core.quit()

        # check if all components have finished
        if not continueRoutine:  # a component has requested a forced-end of Routine
            routineForceEnded = True
            break
        continueRoutine = False  # will revert to True if at least one component still
running
        for thisComponent in Dot_StimuliComponents:
            if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
                continueRoutine = True
                break  # at least one component has not yet finished

        # refresh the screen
        if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
            win.flip()

    # --- Ending Routine "Dot_Stimuli" ---
    for thisComponent in Dot_StimuliComponents:
        if hasattr(thisComponent, "setAutoDraw"):
            thisComponent.setAutoDraw(False)
    # Run 'End Routine' code from code_dot_stimuli
    el_tracker.sendMessage('Dot_Stimuli_Stop')
    #Getting time for when the button is pushed.
    Experiment_Time = expClock.getTime()

    #Getting the reactiontime for excel document
    Reaction_Time = key_reaction.rt

    #Check if participant pressed the right button (1 = right, 0 = wrong)
    if ((Dot_Location == 'left') and (key_reaction.keys == '1')) or ((Dot_Location ==
'right') and (key_reaction.keys == '2')):
        Response_Accuracy = 1
    else:
        Response_Accuracy = 0



    show_dot_left = False
    show_dot_right = False



    #Putting all the info in the list

excel_list.append([Trial_Number,Participant_ID,Face_Stimuli_Left,Face_Stimuli_Rig
ht,Type,Face_Pairs,Gender,Dot_Location,Response_Accuracy,Reaction_Time,Expe
riment_Time, Time_Jitter, VAS])

    if dotStim_congurent.status == STARTED:
        win.callOnFlip(dotStim_congurent.setData, int(0))
    if dotStim_incongurent.status == STARTED:
        win.callOnFlip(dotStim_incongurent.setData, int(0))
    # check responses
    if key_reaction.keys in ['', [], None]:  # No response was made
        key_reaction.keys = None
    thisTrial.addData('key_reaction.keys',key_reaction.keys)
    if key_reaction.keys != None:  # we had a response
        thisTrial.addData('key_reaction.rt', key_reaction.rt)
    # the Routine "Dot_Stimuli" was not non-slip safe, so reset the non-slip timer
    routineTimer.reset()

    # --- Prepare to start Routine "Rest" ---
    continueRoutine = True
    routineForceEnded = False
    # update component parameters for each repeat
    # Run 'Begin Routine' code from code_rest
    el_tracker.sendMessage('Pause_Start')

    Reaction_Time_Display = str(Reaction_Time)[:-11]

    #Making a list for eye tracking messages (used to send messages at a slower
paste)
    et_message_list = []
    et_message_list.append('!V TRIAL_VAR Participant_ID %s' % Participant_ID)
    et_message_list.append('!V TRIAL_VAR Trial_Number %s' % Trial_Number)
    et_message_list.append('!V TRIAL_VAR Face_Stimuli_Left %s' %
Face_Stimuli_Left)
    et_message_list.append('!V TRIAL_VAR Face_Stimuli_Right %s' %
Face_Stimuli_Right)
    et_message_list.append('!V TRIAL_VAR Type %s' % Type)
    et_message_list.append('!V TRIAL_VAR Face_Pairs %s' % Face_Pairs)
    et_message_list.append('!V TRIAL_VAR Gender %s' % Gender)
    et_message_list.append('!V TRIAL_VAR Dot_Location %s' % Dot_Location)
    et_message_list.append('!V TRIAL_VAR Time_Jitter %d' % Time_Jitter)
    et_message_list.append('!V TRIAL_VAR Emotion_Side %s' % Emotion_Side)
    et_message_list.append('!V TRIAL_VAR Reaction_Time %.10f' % Reaction_Time)
    et_message_list.append('!V TRIAL_VAR Response_Accuracy %d' %
Response_Accuracy)


    t2 = 0.1
    counter = 0

    # record trial variables to the EDF data file, for details, see Data
    # Viewer User Manual, "Protocol for EyeLink Data to Viewer Integration"
    #el_tracker.sendMessage('!V TRIAL_VAR Trial_Number %s' % Trial_Number)
    #el_tracker.sendMessage('!V TRIAL_VAR Participant_ID %s' % Participant_ID)
    #el_tracker.sendMessage('!V TRIAL_VAR Face_Stimuli_Left %s' %
Face_Stimuli_Left)
    #el_tracker.sendMessage('!V TRIAL_VAR Face_Stimuli_Right %s' %
Face_Stimuli_Right)
    #el_tracker.sendMessage('!V TRIAL_VAR Type %s' % Type)
    #el_tracker.sendMessage('!V TRIAL_VAR Face_Pairs %s' % Face_Pairs)
    #el_tracker.sendMessage('!V TRIAL_VAR Gender %s' % Gender)
    #el_tracker.sendMessage('!V TRIAL_VAR Dot_Location %s' % Dot_Location)
    #el_tracker.sendMessage('!V TRIAL_VAR Reaction_Time %.10f' %
Reaction_Time)
    #el_tracker.sendMessage('!V TRIAL_VAR Response_Accuracy %d' %
Response_Accuracy)
    reaction_time_numbers.setText(Reaction_Time_Display)
    # keep track of which components have finished
    RestComponents = [reaction, reaction_time_text, reaction_time_numbers]
    for thisComponent in RestComponents:
        thisComponent.tStart = None
        thisComponent.tStop = None
        thisComponent.tStartRefresh = None
        thisComponent.tStopRefresh = None
        if hasattr(thisComponent, 'status'):
            thisComponent.status = NOT_STARTED
    # reset timers
    t = 0
    _timeToFirstFrame = win.getFutureFlipTime(clock="now")
    frameN = -1

    # --- Run Routine "Rest" ---
    while continueRoutine and routineTimer.getTime() < 1.5:
        # get current time
        t = routineTimer.getTime()
```

```
tThisFlip = win.getFutureFlipTime(clock=routineTimer)
tThisFlipGlobal = win.getFutureFlipTime(clock=None)
frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
# update/draw components on each frame
# Run 'Each Frame' code from code_rest
#Timer to send eye tracking messages at a slower paste
t1 = t

if t1 > t2 and counter < 10:
    el_tracker.sendMessage(et_message_list[counter])
    t2 = t + 0.1
    counter += 1



# *reaction* updates
if reaction.status == NOT_STARTED and tThisFlip >= 0.0-frameTolerance:
    # keep track of start time/frame for later
    reaction.frameNStart = frameN  # exact frame index
    reaction.tStart = t  # local t and not account for scr refresh
    reaction.tStartRefresh = tThisFlipGlobal  # on global time
    win.timeOnFlip(reaction, 'tStartRefresh')  # time at next scr refresh
    # add timestamp to datafile
    thisExp.timestampOnFlip(win, 'reaction.started')
    reaction.status = STARTED
    win.callOnFlip(reaction.setData, int(128))
```

```
        if reaction.status == STARTED:
            if frameN >= (reaction.frameNStart + 2.0):
                # keep track of stop time/frame for later
                reaction.tStop = t  # not accounting for scr refresh
                reaction.frameNStop = frameN  # exact frame index
                # add timestamp to datafile
                thisExp.timestampOnFlip(win, 'reaction.stopped')
                reaction.status = FINISHED
                win.callOnFlip(reaction.setData, int(0))

        # *reaction_time_text* updates
        if reaction_time_text.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
            # keep track of start time/frame for later
            reaction_time_text.frameNStart = frameN  # exact frame index
            reaction_time_text.tStart = t  # local t and not account for scr refresh
            reaction_time_text.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(reaction_time_text, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'reaction_time_text.started')
            reaction_time_text.setAutoDraw(True)
        if reaction_time_text.status == STARTED:
            # is it time to stop? (based on global clock, using actual start)
            if tThisFlipGlobal > reaction_time_text.tStartRefresh + 1.5-frameTolerance:
                # keep track of stop time/frame for later
                reaction_time_text.tStop = t  # not accounting for scr refresh
                reaction_time_text.frameNStop = frameN  # exact frame index
                # add timestamp to datafile
                thisExp.timestampOnFlip(win, 'reaction_time_text.stopped')
                reaction_time_text.setAutoDraw(False)

        # *reaction_time_numbers* updates
        if reaction_time_numbers.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
            # keep track of start time/frame for later
            reaction_time_numbers.frameNStart = frameN  # exact frame index
            reaction_time_numbers.tStart = t  # local t and not account for scr refresh
            reaction_time_numbers.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(reaction_time_numbers, 'tStartRefresh')  # time at next scr
refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'reaction_time_numbers.started')
            reaction_time_numbers.setAutoDraw(True)
        if reaction_time_numbers.status == STARTED:
            # is it time to stop? (based on global clock, using actual start)
            if tThisFlipGlobal > reaction_time_numbers.tStartRefresh + 1.5-
frameTolerance:
                # keep track of stop time/frame for later
                reaction_time_numbers.tStop = t  # not accounting for scr refresh
                reaction_time_numbers.frameNStop = frameN  # exact frame index
                # add timestamp to datafile
                thisExp.timestampOnFlip(win, 'reaction_time_numbers.stopped')
                reaction_time_numbers.setAutoDraw(False)
        # check for quit (typically the Esc key)
        if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
            core.quit()

        # check if all components have finished
        if not continueRoutine:  # a component has requested a forced-end of Routine
            routineForceEnded = True
            break
        continueRoutine = False  # will revert to True if at least one component still
running
        for thisComponent in RestComponents:
            if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
                continueRoutine = True
                break  # at least one component has not yet finished

        # refresh the screen
        if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
            win.flip()

    # --- Ending Routine "Rest" ---
    for thisComponent in RestComponents:
        if hasattr(thisComponent, "setAutoDraw"):
            thisComponent.setAutoDraw(False)
    # Run 'End Routine' code from code_rest
    el_tracker.sendMessage('Pause_Stop')
    #el_tracker.sendMessage('Trail_Ended: Reactiontime: ' + str(Reaction_Time) + ', 
Key_Pressed: ' + str(key_resp_1.keys) + ', Response_Accuracy: ' + 
str(Response_Accuracy))

    # record trial variables to the EDF data file, for details, see Data
    # Viewer User Manual, "Protocol for EyeLink Data to Viewer Integration"
    #el_tracker.sendMessage('!V TRIAL_VAR Trial_Number %s' % Trial_Number)
    #el_tracker.sendMessage('!V TRIAL_VAR Participant_ID %s' % Participant_ID)
    #el_tracker.sendMessage('!V TRIAL_VAR Face_Stimuli_Left %s' % 
Face_Stimuli_Left)
    #el_tracker.sendMessage('!V TRIAL_VAR Face_Stimuli_Right %s' % 
Face_Stimuli_Right)
    #el_tracker.sendMessage('!V TRIAL_VAR Type %s' % Type)
    #el_tracker.sendMessage('!V TRIAL_VAR Face_Pairs %s' % Face_Pairs)
    #el_tracker.sendMessage('!V TRIAL_VAR Gender %s' % Gender)
    #el_tracker.sendMessage('!V TRIAL_VAR Dot_Location %s' % Dot_Location)
    #el_tracker.sendMessage('!V TRIAL_VAR Reaction_Time %.10f' % 
Reaction_Time)
    #el_tracker.sendMessage('!V TRIAL_VAR Response_Accuracy %d' % 
Response_Accuracy)

    el_tracker.sendMessage('TRIAL_RESULT %d' % pylink.TRIAL_OK)
    if reaction.status == STARTED:
        win.callOnFlip(reaction.setData, int(0))
    # using non-slip timing so subtract the expected duration of this Routine (unless
ended on request)
    if routineForceEnded:
        routineTimer.reset()
    else:
        routineTimer.addTime(-1.500000)
    thisExp.nextEntry()

# completed 2.0 repeats of 'thisTrial'

# get names of stimulus parameters
if thisTrial.trialList in ([], [None], None):
    params = []
else:
    params = thisTrial.trialList[0].keys()
# save data for this loop
thisTrial.saveAsExcel(filename + '.xlsx', sheetName='thisTrial',
    stimOut=params,
    dataOut=['n','all_mean','all_std', 'all_raw'])
thisTrial.saveAsText(filename + 'thisTrial.csv', delim=',',
    stimOut=params,
    dataOut=['n','all_mean','all_std', 'all_raw'])

# --- Prepare to start Routine "dot_probe_ended_new_experiment_start" ---
continueRoutine = True
routineForceEnded = False
```

```
# update component parameters for each repeat
experiment_phase_2_key.keys = []
experiment_phase_2_key.rt = []
_experiment_phase_2_key_allKeys = []
# keep track of which components have finished
dot_probe_ended_new_experiment_startComponents = [dot_probe_ended_text,
Experiment_ended, experiment_phase_2_key]
for thisComponent in dot_probe_ended_new_experiment_startComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "dot_probe_ended_new_experiment_start" ---
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame


    # *dot_probe_ended_text* updates
    if dot_probe_ended_text.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
        # keep track of start time/frame for later
        dot_probe_ended_text.frameNStart = frameN  # exact frame index
        dot_probe_ended_text.tStart = t  # local t and not account for scr refresh
        dot_probe_ended_text.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(dot_probe_ended_text, 'tStartRefresh')  # time at next scr
refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'dot_probe_ended_text.started')
        dot_probe_ended_text.setAutoDraw(True)
    if dot_probe_ended_text.status == STARTED:
        # is it time to stop? (based on global clock, using actual start)
        if tThisFlipGlobal > dot_probe_ended_text.tStartRefresh + 15-frameTolerance:
            # keep track of stop time/frame for later
            dot_probe_ended_text.tStop = t  # not accounting for scr refresh
            dot_probe_ended_text.frameNStop = frameN  # exact frame index
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'dot_probe_ended_text.stopped')
            dot_probe_ended_text.setAutoDraw(False)
    # *Experiment_ended* updates
    if Experiment_ended.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
        # keep track of start time/frame for later
        Experiment_ended.frameNStart = frameN  # exact frame index
        Experiment_ended.tStart = t  # local t and not account for scr refresh
        Experiment_ended.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(Experiment_ended, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'Experiment_ended.started')
        Experiment_ended.status = STARTED
        win.callOnFlip(Experiment_ended.setData, int(1))
    if Experiment_ended.status == STARTED:
        if frameN >= (Experiment_ended.frameNStart + 2.0):
            # keep track of stop time/frame for later
            Experiment_ended.tStop = t  # not accounting for scr refresh
            Experiment_ended.frameNStop = frameN  # exact frame index
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'Experiment_ended.stopped')
            Experiment_ended.status = FINISHED
            win.callOnFlip(Experiment_ended.setData, int(0))

    # *experiment_phase_2_key* updates
    waitOnFlip = False
    if experiment_phase_2_key.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
        # keep track of start time/frame for later
        experiment_phase_2_key.frameNStart = frameN  # exact frame index
        experiment_phase_2_key.tStart = t  # local t and not account for scr refresh
        experiment_phase_2_key.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(experiment_phase_2_key, 'tStartRefresh')  # time at next scr
refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'experiment_phase_2_key.started')
        experiment_phase_2_key.status = STARTED
        # keyboard checking is just starting
        waitOnFlip = True
        win.callOnFlip(experiment_phase_2_key.clock.reset)  # t=0 on next screen flip
        win.callOnFlip(experiment_phase_2_key.clearEvents, eventType='keyboard')  #
clear events on next screen flip
    if experiment_phase_2_key.status == STARTED and not waitOnFlip:
        theseKeys = experiment_phase_2_key.getKeys(keyList=['1','2'],
waitRelease=False)
        _experiment_phase_2_key_allKeys.extend(theseKeys)
        if len(_experiment_phase_2_key_allKeys):
            experiment_phase_2_key.keys = _experiment_phase_2_key_allKeys[-
1].name  # just the last key pressed
            experiment_phase_2_key.rt = _experiment_phase_2_key_allKeys[-1].rt
            # a response ends the routine
            continueRoutine = False

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
```

```
            routineForceEnded = True
            break
        continueRoutine = False  # will revert to True if at least one component still running
        for thisComponent in dot_probe_ended_new_experiment_startComponents:
            if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
                continueRoutine = True
                break  # at least one component has not yet finished

        # refresh the screen
        if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
            win.flip()

    # --- Ending Routine "dot_probe_ended_new_experiment_start" ---
    for thisComponent in dot_probe_ended_new_experiment_startComponents:
        if hasattr(thisComponent, "setAutoDraw"):
            thisComponent.setAutoDraw(False)
    # Run 'End Routine' code from end_dot_probe_code
    #Saves the info from dot-probe experment list in a excel document
    np.savetxt(Participant_ID + "_" + expName + "_" + expInfo['date'] + ".csv", excel_list,
    delimiter = ",", fmt ='% s')

    el_tracker.sendMessage('Start_Experiment_Phase_2')



    if Experiment_ended.status == STARTED:
        win.callOnFlip(Experiment_ended.setData, int(0))
```

```
# check responses
if experiment_phase_2_key.keys in ['', [], None]:  # No response was made
    experiment_phase_2_key.keys = None
thisExp.addData('experiment_phase_2_key.keys',experiment_phase_2_key.keys)
if experiment_phase_2_key.keys != None:  # we had a response
    thisExp.addData('experiment_phase_2_key.rt', experiment_phase_2_key.rt)
thisExp.nextEntry()
# the Routine "dot_probe_ended_new_experiment_start" was not non-slip safe, so
reset the non-slip timer
routineTimer.reset()

# --- Prepare to start Routine "smooth_pursuit_info" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
smooth_pursuit_info_keyboard.keys = []
smooth_pursuit_info_keyboard.rt = []
_smooth_pursuit_info_keyboard_allKeys = []
# keep track of which components have finished
smooth_pursuit_infoComponents = [smooth_pursuit_info_text,
smooth_pursuit_info_keyboard]
for thisComponent in smooth_pursuit_infoComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "smooth_pursuit_info" ---
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # *smooth_pursuit_info_text* updates
    if smooth_pursuit_info_text.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
        # keep track of start time/frame for later
        smooth_pursuit_info_text.frameNStart = frameN  # exact frame index
        smooth_pursuit_info_text.tStart = t  # local t and not account for scr refresh
        smooth_pursuit_info_text.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(smooth_pursuit_info_text, 'tStartRefresh')  # time at next scr
refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'smooth_pursuit_info_text.started')
        smooth_pursuit_info_text.setAutoDraw(True)
    if smooth_pursuit_info_text.status == STARTED:
        # is it time to stop? (based on global clock, using actual start)
        if tThisFlipGlobal > smooth_pursuit_info_text.tStartRefresh + 15-
frameTolerance:
            # keep track of stop time/frame for later
            smooth_pursuit_info_text.tStop = t  # not accounting for scr refresh
            smooth_pursuit_info_text.frameNStop = frameN  # exact frame index
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'smooth_pursuit_info_text.stopped')
            smooth_pursuit_info_text.setAutoDraw(False)

    # *smooth_pursuit_info_keyboard* updates
    waitOnFlip = False
    if smooth_pursuit_info_keyboard.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
        # keep track of start time/frame for later
        smooth_pursuit_info_keyboard.frameNStart = frameN  # exact frame index
        smooth_pursuit_info_keyboard.tStart = t  # local t and not account for scr
refresh
        smooth_pursuit_info_keyboard.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(smooth_pursuit_info_keyboard, 'tStartRefresh')  # time at next
scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'smooth_pursuit_info_keyboard.started')
        smooth_pursuit_info_keyboard.status = STARTED
        # keyboard checking is just starting
        waitOnFlip = True
        win.callOnFlip(smooth_pursuit_info_keyboard.clock.reset)  # t=0 on next screen
flip
        win.callOnFlip(smooth_pursuit_info_keyboard.clearEvents,
eventType='keyboard')  # clear events on next screen flip
    if smooth_pursuit_info_keyboard.status == STARTED and not waitOnFlip:
        theseKeys = smooth_pursuit_info_keyboard.getKeys(keyList=['1','2'],
waitRelease=False)
        _smooth_pursuit_info_keyboard_allKeys.extend(theseKeys)
        if len(_smooth_pursuit_info_keyboard_allKeys):
            smooth_pursuit_info_keyboard.keys =
_smooth_pursuit_info_keyboard_allKeys[-1].name  # just the last key pressed
            smooth_pursuit_info_keyboard.rt = _smooth_pursuit_info_keyboard_allKeys[-
1].rt
            # a response ends the routine
            continueRoutine = False

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in smooth_pursuit_infoComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished

    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()

# --- Ending Routine "smooth_pursuit_info" ---
for thisComponent in smooth_pursuit_infoComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
# check responses
if smooth_pursuit_info_keyboard.keys in ['', [], None]:  # No response was made
    smooth_pursuit_info_keyboard.keys = None
thisExp.addData('smooth_pursuit_info_keyboard.keys',smooth_pursuit_info_keyboar
d.keys)
if smooth_pursuit_info_keyboard.keys != None:  # we had a response
    thisExp.addData('smooth_pursuit_info_keyboard.rt',
smooth_pursuit_info_keyboard.rt)
thisExp.nextEntry()
# the Routine "smooth_pursuit_info" was not non-slip safe, so reset the non-slip timer
routineTimer.reset()

# --- Prepare to start Routine "smooth_pursuit" ---
```

```
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
# Run 'Begin Routine' code from code_4
el_tracker.sendMessage('smooth_pursuit_start')
import math
x=0
y=0
# keep track of which components have finished
smooth_pursuitComponents = [polygon_smooth]
for thisComponent in smooth_pursuitComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "smooth_pursuit" ---
while continueRoutine and routineTimer.getTime() < 45.0:
    # get current time
    t = routineTimer.getTime()
```

```
tThisFlip = win.getFutureFlipTime(clock=routineTimer)
tThisFlipGlobal = win.getFutureFlipTime(clock=None)
frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
# update/draw components on each frame

# *polygon_smooth* updates
if polygon_smooth.status == NOT_STARTED and tThisFlip >= 0-frameTolerance:
    # keep track of start time/frame for later
    polygon_smooth.frameNStart = frameN  # exact frame index
    polygon_smooth.tStart = t  # local t and not account for scr refresh
    polygon_smooth.tStartRefresh = tThisFlipGlobal  # on global time
    win.timeOnFlip(polygon_smooth, 'tStartRefresh')  # time at next scr refresh
    # add timestamp to datafile
    thisExp.timestampOnFlip(win, 'polygon_smooth.started')
    polygon_smooth.setAutoDraw(True)
if polygon_smooth.status == STARTED:
    # is it time to stop? (based on global clock, using actual start)
    if tThisFlipGlobal > polygon_smooth.tStartRefresh + 45-frameTolerance:
        # keep track of stop time/frame for later
        polygon_smooth.tStop = t  # not accounting for scr refresh
        polygon_smooth.frameNStop = frameN  # exact frame index
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'polygon_smooth.stopped')
        polygon_smooth.setAutoDraw(False)
# Run 'Each Frame' code from code_4
x = 0.5*math.sin(0.04*frameN)
```

```
    y = 0.3*math.sin(0.05*frameN)

    polygon_smooth.pos = (x,y)

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in smooth_pursuitComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished

    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()

# --- Ending Routine "smooth_pursuit" ---
for thisComponent in smooth_pursuitComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
```

```
# Run 'End Routine' code from code_4
el_tracker.sendMessage('smooth_pursuit_end')
# using non-slip timing so subtract the expected duration of this Routine (unless
ended on request)
if routineForceEnded:
    routineTimer.reset()
else:
    routineTimer.addTime(-45.000000)

# set up handler to look after randomisation of conditions etc
saccade_trial = data.TrialHandler(nReps=2.0, method='sequential',
    extraInfo=expInfo, originPath=-1,
    trialList=[None],
    seed=None, name='saccade_trial')
thisExp.addLoop(saccade_trial)  # add the loop to the experiment
thisSaccade_trial = saccade_trial.trialList[0]  # so we can initialise stimuli with some
values
# abbreviate parameter names if possible (e.g. rgb = thisSaccade_trial.rgb)
if thisSaccade_trial != None:
    for paramName in thisSaccade_trial:
        exec('{} = thisSaccade_trial[paramName]'.format(paramName))

for thisSaccade_trial in saccade_trial:
    currentLoop = saccade_trial
    # abbreviate parameter names if possible (e.g. rgb = thisSaccade_trial.rgb)
    if thisSaccade_trial != None:
```

```
    for paramName in thisSaccade_trial:
        exec('{} = thisSaccade_trial[paramName]'.format(paramName))

# --- Prepare to start Routine "saccade" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
# Run 'Begin Routine' code from code_5
el_tracker.sendMessage('saccade_start')
import math
import numpy as np
import psychopy.clock

x = 1
y = 0
frame_i = frameN
frames_p_s = 30
frames_peri = 1.0/frames_p_s

curr_time = 0
curr_index = 0

x0 = 0.5
y0= 0.3
```

```
    times = [0,1.2078598 , 2.19513117, 1.90718612, 1.85591199, 1.82492585,
           1.64140316, 2.3855269 , 2.08878815, 1.62005794, 1.57281463,
           1.75342754, 1.71885385, 1.53225212, 2.19249282, 2.64586055,
           1.31651686, 2.87176734, 1.79632565, 2.10867142, 2.11795502,
           1.5877191 , 2.07060512, 1.69428916, 1.83312158, 2.6838467,10000 ]

    cumtimes = np.cumsum(times)
    tot_time = np.sum(times)
    #gpositions = [(0, 0),(-1, 0), (0, 0), (-1, 1),(0, 0), (1, 1),(0, 0), (0, -1), (0, 0), (1, -
1),(0, 0), (-1, -1), (0, 0),(0, 1),(0, 0), (1, 0),(1, -1), (0, 0), (-1, 1), (1, 0), (1, 1), (0, -1),
(0, 1), (-1, -1), (-1, 0)]
    gpositions = [(0*x0, 0),(0*x0, 0),(-1*x0, 0), (0*x0, 0), (-1*x0, 1*y0),(0*x0, 0), (1*x0,
1*y0),(0, 0), (0, -1*y0), (0, 0), (1*x0, -1*y0),(0, 0), (-1*x0, -1*y0), (0, 0),(0, 1*y0),(0, 0),
(1*x0, 0),(1*x0, -1*y0), (0, 0), (-1*x0, 1*y0), (1*x0, 0), (1*x0, 1*y0), (0, -1*y0), (0,
1*y0), (-1*x0, -1*y0), (-1*x0, 0),(0, 0)]
    timer = core.Clock()
    timer.add(np.sum(times))
    # keep track of which components have finished
    saccadeComponents = [polygon_saccade]
    for thisComponent in saccadeComponents:
        thisComponent.tStart = None
        thisComponent.tStop = None
        thisComponent.tStartRefresh = None
        thisComponent.tStopRefresh = None
        if hasattr(thisComponent, 'status'):
            thisComponent.status = NOT_STARTED
```

```
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "saccade" ---
while continueRoutine and routineTimer.getTime() < 25.0:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame
    # Run 'Each Frame' code from code_5

    curr_time += frames_peri

    if curr_time>cumtimes[curr_index]:
        curr_index +=1
        curr_pos = gpositions[curr_index]



    polygon_saccade.pos = curr_pos



    # *polygon_saccade* updates
```

```
        if polygon_saccade.status == NOT_STARTED and tThisFlip >= 0-
frameTolerance:
            # keep track of start time/frame for later
            polygon_saccade.frameNStart = frameN  # exact frame index
            polygon_saccade.tStart = t  # local t and not account for scr refresh
            polygon_saccade.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(polygon_saccade, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'polygon_saccade.started')
            polygon_saccade.setAutoDraw(True)
        if polygon_saccade.status == STARTED:
            # is it time to stop? (based on global clock, using actual start)
            if tThisFlipGlobal > polygon_saccade.tStartRefresh + 25-frameTolerance:
                # keep track of stop time/frame for later
                polygon_saccade.tStop = t  # not accounting for scr refresh
                polygon_saccade.frameNStop = frameN  # exact frame index
                # add timestamp to datafile
                thisExp.timestampOnFlip(win, 'polygon_saccade.stopped')
                polygon_saccade.setAutoDraw(False)

        # check for quit (typically the Esc key)
        if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
            core.quit()

        # check if all components have finished
        if not continueRoutine:  # a component has requested a forced-end of Routine
```

```
            routineForceEnded = True
            break
        continueRoutine = False  # will revert to True if at least one component still
running
        for thisComponent in saccadeComponents:
            if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
                continueRoutine = True
                break  # at least one component has not yet finished

        # refresh the screen
        if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
            win.flip()

    # --- Ending Routine "saccade" ---
    for thisComponent in saccadeComponents:
        if hasattr(thisComponent, "setAutoDraw"):
            thisComponent.setAutoDraw(False)
    # Run 'End Routine' code from code_5
    el_tracker.sendMessage('saccade_end')
    # using non-slip timing so subtract the expected duration of this Routine (unless
ended on request)
    if routineForceEnded:
        routineTimer.reset()
    else:
        routineTimer.addTime(-25.000000)
    thisExp.nextEntry()
```

```
# completed 2.0 repeats of 'saccade_trial'


# get names of stimulus parameters
if saccade_trial.trialList in ([], [None], None):
    params = []
else:
    params = saccade_trial.trialList[0].keys()
# save data for this loop
saccade_trial.saveAsExcel(filename + '.xlsx', sheetName='saccade_trial',
    stimOut=params,
    dataOut=['n','all_mean','all_std', 'all_raw'])
saccade_trial.saveAsText(filename + 'saccade_trial.csv', delim=',',
    stimOut=params,
    dataOut=['n','all_mean','all_std', 'all_raw'])


# --- Prepare to start Routine "intro_waldo" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
intro_waldo_keyboard.keys = []
intro_waldo_keyboard.rt = []
_intro_waldo_keyboard_allKeys = []
# keep track of which components have finished
intro_waldoComponents = [intro_waldo_picture, intro_waldo_keyboard]
for thisComponent in intro_waldoComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "intro_waldo" ---
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # *intro_waldo_picture* updates
    if intro_waldo_picture.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
        # keep track of start time/frame for later
        intro_waldo_picture.frameNStart = frameN  # exact frame index
        intro_waldo_picture.tStart = t  # local t and not account for scr refresh
        intro_waldo_picture.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(intro_waldo_picture, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'intro_waldo_picture.started')
        intro_waldo_picture.setAutoDraw(True)

    # *intro_waldo_keyboard* updates
    waitOnFlip = False
    if intro_waldo_keyboard.status == NOT_STARTED and tThisFlip >= 0.0-
frameTolerance:
        # keep track of start time/frame for later
        intro_waldo_keyboard.frameNStart = frameN  # exact frame index
        intro_waldo_keyboard.tStart = t  # local t and not account for scr refresh
        intro_waldo_keyboard.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(intro_waldo_keyboard, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'intro_waldo_keyboard.started')
        intro_waldo_keyboard.status = STARTED
        # keyboard checking is just starting
        waitOnFlip = True
        win.callOnFlip(intro_waldo_keyboard.clock.reset)  # t=0 on next screen flip
        win.callOnFlip(intro_waldo_keyboard.clearEvents, eventType='keyboard')  #
clear events on next screen flip
    if intro_waldo_keyboard.status == STARTED and not waitOnFlip:
        theseKeys = intro_waldo_keyboard.getKeys(keyList=['1','2'],
waitRelease=False)
        _intro_waldo_keyboard_allKeys.extend(theseKeys)
        if len(_intro_waldo_keyboard_allKeys):
            intro_waldo_keyboard.keys = _intro_waldo_keyboard_allKeys[-1].name  #
just the last key pressed
            intro_waldo_keyboard.rt = _intro_waldo_keyboard_allKeys[-1].rt
            # a response ends the routine
            continueRoutine = False

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in intro_waldoComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished

    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()

# --- Ending Routine "intro_waldo" ---
```

```
for thisComponent in intro_waldoComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
# check responses
if intro_waldo_keyboard.keys in ['', [], None]:  # No response was made
    intro_waldo_keyboard.keys = None
thisExp.addData('intro_waldo_keyboard.keys',intro_waldo_keyboard.keys)
if intro_waldo_keyboard.keys != None:  # we had a response
    thisExp.addData('intro_waldo_keyboard.rt', intro_waldo_keyboard.rt)
thisExp.nextEntry()
# the Routine "intro_waldo" was not non-slip safe, so reset the non-slip timer
routineTimer.reset()

# --- Prepare to start Routine "waldo" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
# Run 'Begin Routine' code from code_6
el_tracker.sendMessage('waldo_experiment_start')
# keep track of which components have finished
waldoComponents = [image, image_2, image_3, image_4, image_5, image_6]
for thisComponent in waldoComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "waldo" ---
while continueRoutine and routineTimer.getTime() < 270.0:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # *image* updates
    if image.status == NOT_STARTED and tThisFlip >= 0.0-frameTolerance:
        # keep track of start time/frame for later
        image.frameNStart = frameN  # exact frame index
        image.tStart = t  # local t and not account for scr refresh
        image.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(image, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'image.started')
        image.setAutoDraw(True)
```

```
if image.status == STARTED:
    # is it time to stop? (based on global clock, using actual start)
    if tThisFlipGlobal > image.tStartRefresh + 45-frameTolerance:
        # keep track of stop time/frame for later
        image.tStop = t  # not accounting for scr refresh
        image.frameNStop = frameN  # exact frame index
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'image.stopped')
        image.setAutoDraw(False)

# *image_2* updates
if image_2.status == NOT_STARTED and tThisFlip >= 45-frameTolerance:
    # keep track of start time/frame for later
    image_2.frameNStart = frameN  # exact frame index
    image_2.tStart = t  # local t and not account for scr refresh
    image_2.tStartRefresh = tThisFlipGlobal  # on global time
    win.timeOnFlip(image_2, 'tStartRefresh')  # time at next scr refresh
    # add timestamp to datafile
    thisExp.timestampOnFlip(win, 'image_2.started')
    image_2.setAutoDraw(True)
if image_2.status == STARTED:
    # is it time to stop? (based on global clock, using actual start)
    if tThisFlipGlobal > image_2.tStartRefresh + 45-frameTolerance:
        # keep track of stop time/frame for later
        image_2.tStop = t  # not accounting for scr refresh
        image_2.frameNStop = frameN  # exact frame index
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'image_2.stopped')
        image_2.setAutoDraw(False)

# *image_3* updates
if image_3.status == NOT_STARTED and tThisFlip >= 90-frameTolerance:
    # keep track of start time/frame for later
    image_3.frameNStart = frameN  # exact frame index
    image_3.tStart = t  # local t and not account for scr refresh
    image_3.tStartRefresh = tThisFlipGlobal  # on global time
    win.timeOnFlip(image_3, 'tStartRefresh')  # time at next scr refresh
    # add timestamp to datafile
    thisExp.timestampOnFlip(win, 'image_3.started')
    image_3.setAutoDraw(True)
if image_3.status == STARTED:
    # is it time to stop? (based on global clock, using actual start)
    if tThisFlipGlobal > image_3.tStartRefresh + 45-frameTolerance:
        # keep track of stop time/frame for later
        image_3.tStop = t  # not accounting for scr refresh
        image_3.frameNStop = frameN  # exact frame index
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'image_3.stopped')
        image_3.setAutoDraw(False)

# *image_4* updates
if image_4.status == NOT_STARTED and tThisFlip >= 135-frameTolerance:
    # keep track of start time/frame for later
    image_4.frameNStart = frameN  # exact frame index
    image_4.tStart = t  # local t and not account for scr refresh
    image_4.tStartRefresh = tThisFlipGlobal  # on global time
    win.timeOnFlip(image_4, 'tStartRefresh')  # time at next scr refresh
    # add timestamp to datafile
    thisExp.timestampOnFlip(win, 'image_4.started')
    image_4.setAutoDraw(True)
if image_4.status == STARTED:
    # is it time to stop? (based on global clock, using actual start)
    if tThisFlipGlobal > image_4.tStartRefresh + 45-frameTolerance:
        # keep track of stop time/frame for later
        image_4.tStop = t  # not accounting for scr refresh
        image_4.frameNStop = frameN  # exact frame index
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'image_4.stopped')
        image_4.setAutoDraw(False)

# *image_5* updates
if image_5.status == NOT_STARTED and tThisFlip >= 180-frameTolerance:
    # keep track of start time/frame for later
    image_5.frameNStart = frameN  # exact frame index
    image_5.tStart = t  # local t and not account for scr refresh
    image_5.tStartRefresh = tThisFlipGlobal  # on global time
    win.timeOnFlip(image_5, 'tStartRefresh')  # time at next scr refresh
    # add timestamp to datafile
    thisExp.timestampOnFlip(win, 'image_5.started')
    image_5.setAutoDraw(True)
if image_5.status == STARTED:
    # is it time to stop? (based on global clock, using actual start)
    if tThisFlipGlobal > image_5.tStartRefresh + 45-frameTolerance:
        # keep track of stop time/frame for later
        image_5.tStop = t  # not accounting for scr refresh
        image_5.frameNStop = frameN  # exact frame index
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'image_5.stopped')
        image_5.setAutoDraw(False)

# *image_6* updates
if image_6.status == NOT_STARTED and tThisFlip >= 225-frameTolerance:
    # keep track of start time/frame for later
    image_6.frameNStart = frameN  # exact frame index
    image_6.tStart = t  # local t and not account for scr refresh
    image_6.tStartRefresh = tThisFlipGlobal  # on global time
    win.timeOnFlip(image_6, 'tStartRefresh')  # time at next scr refresh
    # add timestamp to datafile
    thisExp.timestampOnFlip(win, 'image_6.started')
    image_6.setAutoDraw(True)
if image_6.status == STARTED:
    # is it time to stop? (based on global clock, using actual start)
    if tThisFlipGlobal > image_6.tStartRefresh + 45-frameTolerance:
        # keep track of stop time/frame for later
        image_6.tStop = t  # not accounting for scr refresh
        image_6.frameNStop = frameN  # exact frame index
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'image_6.stopped')
        image_6.setAutoDraw(False)
# Run 'Each Frame' code from code_6
count = 0
if int(t / 45) > count:
    el_tracker.sendMessage('waldo_picture_'+str(count))
    count = count + 1
    el_tracker.sendMessage('waldo_experiment_change_pic = '+str(count))



# check for quit (typically the Esc key)
if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
    core.quit()

# check if all components have finished
if not continueRoutine:  # a component has requested a forced-end of Routine
    routineForceEnded = True
    break
continueRoutine = False  # will revert to True if at least one component still running
for thisComponent in waldoComponents:
    if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
        continueRoutine = True
        break  # at least one component has not yet finished
    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()

# --- Ending Routine "waldo" ---
for thisComponent in waldoComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
# Run 'End Routine' code from code_6
el_tracker.sendMessage('waldo_experiment_end')
# using non-slip timing so subtract the expected duration of this Routine (unless
ended on request)
if routineForceEnded:
    routineTimer.reset()
else:
    routineTimer.addTime(-270.000000)

# --- Prepare to start Routine "intro_rand_pix" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
key_resp_2.keys = []
key_resp_2.rt = []
_key_resp_2_allKeys = []
# keep track of which components have finished
```

```
intro_rand_pixComponents = [text_3, key_resp_2]
for thisComponent in intro_rand_pixComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "intro_rand_pix" ---
while continueRoutine:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # *text_3* updates
    if text_3.status == NOT_STARTED and tThisFlip >= 0.0-frameTolerance:
        # keep track of start time/frame for later
        text_3.frameNStart = frameN  # exact frame index
```

```
            text_3.tStart = t  # local t and not account for scr refresh
            text_3.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(text_3, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'text_3.started')
            text_3.setAutoDraw(True)

        # *key_resp_2* updates
        waitOnFlip = False
        if key_resp_2.status == NOT_STARTED and tThisFlip >= 0.0-frameTolerance:
            # keep track of start time/frame for later
            key_resp_2.frameNStart = frameN  # exact frame index
            key_resp_2.tStart = t  # local t and not account for scr refresh
            key_resp_2.tStartRefresh = tThisFlipGlobal  # on global time
            win.timeOnFlip(key_resp_2, 'tStartRefresh')  # time at next scr refresh
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'key_resp_2.started')
            key_resp_2.status = STARTED
            # keyboard checking is just starting
            waitOnFlip = True
            win.callOnFlip(key_resp_2.clock.reset)  # t=0 on next screen flip
            win.callOnFlip(key_resp_2.clearEvents, eventType='keyboard')  # clear events
    on next screen flip
        if key_resp_2.status == STARTED and not waitOnFlip:
            theseKeys = key_resp_2.getKeys(keyList=['1','2'], waitRelease=False)
            _key_resp_2_allKeys.extend(theseKeys)
            if len(_key_resp_2_allKeys):
                key_resp_2.keys = _key_resp_2_allKeys[-1].name  # just the last key
    pressed
                key_resp_2.rt = _key_resp_2_allKeys[-1].rt
                # a response ends the routine
                continueRoutine = False

        # check for quit (typically the Esc key)
        if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
            core.quit()

        # check if all components have finished
        if not continueRoutine:  # a component has requested a forced-end of Routine
            routineForceEnded = True
            break
        continueRoutine = False  # will revert to True if at least one component still running
        for thisComponent in intro_rand_pixComponents:
            if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
                continueRoutine = True
                break  # at least one component has not yet finished

        # refresh the screen
        if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
            win.flip()

    # --- Ending Routine "intro_rand_pix" ---
```

```
for thisComponent in intro_rand_pixComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
# check responses
if key_resp_2.keys in ['', [], None]:  # No response was made
    key_resp_2.keys = None
thisExp.addData('key_resp_2.keys',key_resp_2.keys)
if key_resp_2.keys != None:  # we had a response
    thisExp.addData('key_resp_2.rt', key_resp_2.rt)
thisExp.nextEntry()
# the Routine "intro_rand_pix" was not non-slip safe, so reset the non-slip timer
routineTimer.reset()

# --- Prepare to start Routine "rand_pix" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
# Run 'Begin Routine' code from code_7
el_tracker.sendMessage('start_rand_pix')
# keep track of which components have finished
rand_pixComponents = [image_7, image_8, image_9, image_10]
for thisComponent in rand_pixComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1

# --- Run Routine "rand_pix" ---
while continueRoutine and routineTimer.getTime() < 120.0:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # *image_7* updates
    if image_7.status == NOT_STARTED and tThisFlip >= 0-frameTolerance:
        # keep track of start time/frame for later
        image_7.frameNStart = frameN  # exact frame index
        image_7.tStart = t  # local t and not account for scr refresh
        image_7.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(image_7, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'image_7.started')
        image_7.setAutoDraw(True)
```

```
if image_7.status == STARTED:
    # is it time to stop? (based on global clock, using actual start)
    if tThisFlipGlobal > image_7.tStartRefresh + 30-frameTolerance:
        # keep track of stop time/frame for later
        image_7.tStop = t  # not accounting for scr refresh
        image_7.frameNStop = frameN  # exact frame index
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'image_7.stopped')
        image_7.setAutoDraw(False)

# *image_8* updates
if image_8.status == NOT_STARTED and tThisFlip >= 30-frameTolerance:
    # keep track of start time/frame for later
    image_8.frameNStart = frameN  # exact frame index
    image_8.tStart = t  # local t and not account for scr refresh
    image_8.tStartRefresh = tThisFlipGlobal  # on global time
    win.timeOnFlip(image_8, 'tStartRefresh')  # time at next scr refresh
    # add timestamp to datafile
    thisExp.timestampOnFlip(win, 'image_8.started')
    image_8.setAutoDraw(True)
if image_8.status == STARTED:
    # is it time to stop? (based on global clock, using actual start)
    if tThisFlipGlobal > image_8.tStartRefresh + 30-frameTolerance:
        # keep track of stop time/frame for later
        image_8.tStop = t  # not accounting for scr refresh
        image_8.frameNStop = frameN  # exact frame index
```

```
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'image_8.stopped')
            image_8.setAutoDraw(False)

    # *image_9* updates
    if image_9.status == NOT_STARTED and tThisFlip >= 60-frameTolerance:
        # keep track of start time/frame for later
        image_9.frameNStart = frameN  # exact frame index
        image_9.tStart = t  # local t and not account for scr refresh
        image_9.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(image_9, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'image_9.started')
        image_9.setAutoDraw(True)
    if image_9.status == STARTED:
        # is it time to stop? (based on global clock, using actual start)
        if tThisFlipGlobal > image_9.tStartRefresh + 30-frameTolerance:
            # keep track of stop time/frame for later
            image_9.tStop = t  # not accounting for scr refresh
            image_9.frameNStop = frameN  # exact frame index
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'image_9.stopped')
            image_9.setAutoDraw(False)

    # *image_10* updates
    if image_10.status == NOT_STARTED and tThisFlip >= 90-frameTolerance:
        # keep track of start time/frame for later
        image_10.frameNStart = frameN  # exact frame index
        image_10.tStart = t  # local t and not account for scr refresh
        image_10.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(image_10, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'image_10.started')
        image_10.setAutoDraw(True)
    if image_10.status == STARTED:
        # is it time to stop? (based on global clock, using actual start)
        if tThisFlipGlobal > image_10.tStartRefresh + 30-frameTolerance:
            # keep track of stop time/frame for later
            image_10.tStop = t  # not accounting for scr refresh
            image_10.frameNStop = frameN  # exact frame index
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'image_10.stopped')
            image_10.setAutoDraw(False)
    # Run 'Each Frame' code from code_7
    count = 0
    if int(t / 45) > count:
        el_tracker.sendMessage('rand_pix'+str(count))
        count = count + 1
        el_tracker.sendMessage('rand_pix_pic = '+str(count))

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
            core.quit()

        # check if all components have finished
        if not continueRoutine:  # a component has requested a forced-end of Routine
            routineForceEnded = True
            break
        continueRoutine = False  # will revert to True if at least one component still running
        for thisComponent in rand_pixComponents:
            if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
                continueRoutine = True
                break  # at least one component has not yet finished

        # refresh the screen
        if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
            win.flip()

    # --- Ending Routine "rand_pix" ---
    for thisComponent in rand_pixComponents:
        if hasattr(thisComponent, "setAutoDraw"):
            thisComponent.setAutoDraw(False)
    # Run 'End Routine' code from code_7
    el_tracker.sendMessage('end_rand_pix')
    # using non-slip timing so subtract the expected duration of this Routine (unless
ended on request)
    if routineForceEnded:
        routineTimer.reset()
```

```
else:
    routineTimer.addTime(-120.000000)

# --- Prepare to start Routine "End" ---
continueRoutine = True
routineForceEnded = False
# update component parameters for each repeat
# Run 'Begin Routine' code from experiment_over_code
el_tracker.sendMessage('endExperiment')

el_tracker.stopRecording()



# keep track of which components have finished
EndComponents = [End_text]
for thisComponent in EndComponents:
    thisComponent.tStart = None
    thisComponent.tStop = None
    thisComponent.tStartRefresh = None
    thisComponent.tStopRefresh = None
    if hasattr(thisComponent, 'status'):
        thisComponent.status = NOT_STARTED
# reset timers
t = 0
_timeToFirstFrame = win.getFutureFlipTime(clock="now")
frameN = -1
```

```
# --- Run Routine "End" ---
while continueRoutine and routineTimer.getTime() < 5.0:
    # get current time
    t = routineTimer.getTime()
    tThisFlip = win.getFutureFlipTime(clock=routineTimer)
    tThisFlipGlobal = win.getFutureFlipTime(clock=None)
    frameN = frameN + 1  # number of completed frames (so 0 is the first frame)
    # update/draw components on each frame

    # *End_text* updates
    if End_text.status == NOT_STARTED and tThisFlip >= 0.0-frameTolerance:
        # keep track of start time/frame for later
        End_text.frameNStart = frameN  # exact frame index
        End_text.tStart = t  # local t and not account for scr refresh
        End_text.tStartRefresh = tThisFlipGlobal  # on global time
        win.timeOnFlip(End_text, 'tStartRefresh')  # time at next scr refresh
        # add timestamp to datafile
        thisExp.timestampOnFlip(win, 'End_text.started')
        End_text.setAutoDraw(True)
    if End_text.status == STARTED:
        # is it time to stop? (based on global clock, using actual start)
        if tThisFlipGlobal > End_text.tStartRefresh + 5-frameTolerance:
            # keep track of stop time/frame for later
            End_text.tStop = t  # not accounting for scr refresh
            End_text.frameNStop = frameN  # exact frame index
            # add timestamp to datafile
            thisExp.timestampOnFlip(win, 'End_text.stopped')
            End_text.setAutoDraw(False)

    # check for quit (typically the Esc key)
    if endExpNow or defaultKeyboard.getKeys(keyList=["escape"]):
        core.quit()

    # check if all components have finished
    if not continueRoutine:  # a component has requested a forced-end of Routine
        routineForceEnded = True
        break
    continueRoutine = False  # will revert to True if at least one component still running
    for thisComponent in EndComponents:
        if hasattr(thisComponent, "status") and thisComponent.status != FINISHED:
            continueRoutine = True
            break  # at least one component has not yet finished

    # refresh the screen
    if continueRoutine:  # don't flip if this routine is over or we'll get a blank screen
        win.flip()

# --- Ending Routine "End" ---
for thisComponent in EndComponents:
    if hasattr(thisComponent, "setAutoDraw"):
        thisComponent.setAutoDraw(False)
```

```
# using non-slip timing so subtract the expected duration of this Routine (unless
ended on request)
if routineForceEnded:
    routineTimer.reset()
else:
    routineTimer.addTime(-5.000000)
# Run 'End Experiment' code from Start_and_end_code
# Step 7: disconnect, download the EDF file, then terminate the task
terminate_task()




# --- End experiment ---
# Flip one final time so any remaining win.callOnFlip()
# and win.timeOnFlip() tasks get executed before quitting
win.flip()

# these shouldn't be strictly necessary (should auto-save)
thisExp.saveAsPickle(filename)
logging.flush()
# make sure everything is closed down
if eyetracker:
    eyetracker.setConnectionState(False)
thisExp.abort()  # or data files will save again on exit
```

```
win.close()

core.quit()
```