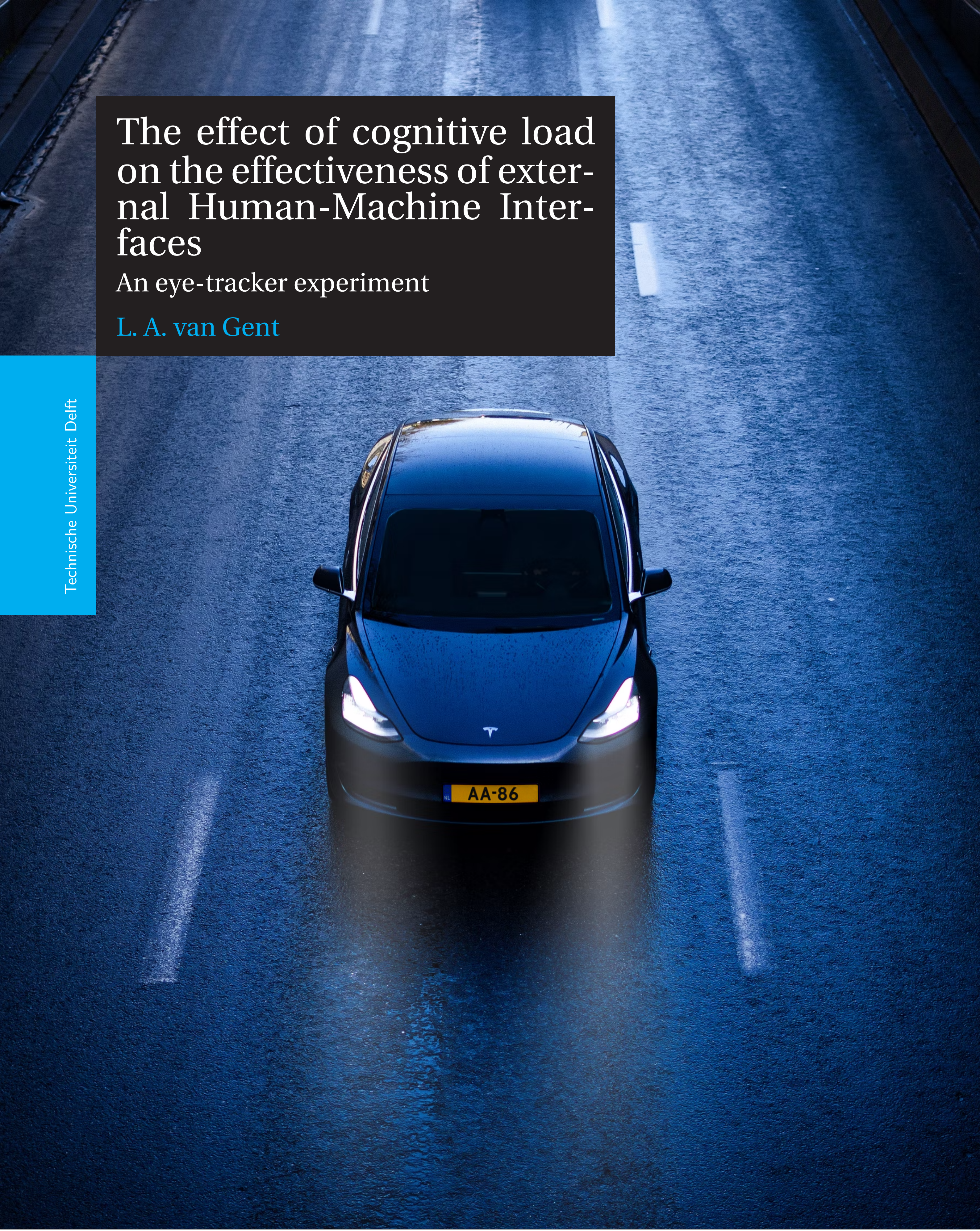


The effect of cognitive load on the effectiveness of external Human-Machine Interfaces

An eye-tracker experiment

L. A. van Gent

THE EFFECT OF COGNITIVE LOAD ON THE EFFECTIVENESS OF EXTERNAL HUMAN-MACHINE INTERFACES

AN EYE-TRACKER EXPERIMENT

by

L. A. van Gent

in partial fulfillment of the requirements for the degree of

Master of Science
in Mechanical Engineering

at the Delft University of Technology,
to be defended publicly on Monday July 25th, 2022 at 10:00 AM.

Supervisor:	Dr. Ir. Y. B. Eisma,	TU Delft (3mE-CoR)
	Dr. Ir. J. C. F. de Winter,	TU Delft (3mE-CoR)
Thesis committee:	Dr. Ir. Y. B. Eisma,	TU Delft (3mE-CoR)
	Dr. Ir. J. C. F. de Winter,	TU Delft (3mE-CoR)
	Dr. D. Dodou,	TU Delft (3mE-BMechE)

Abstract

With the increase in development of self-driving cars, research has been conducted to retain humanized interaction between cars and other road users, such as pedestrians. One way to retain this type of interaction is through the use of external Human-Machine Interfaces (eHMIs). This project aims to contribute to this field of research by exploring the case of eHMI-equipped, self-driving cars yielding for a crossing pedestrian. From literature it is known that light- and text-based signals are both promising ways of utilizing an eHMI. Due to the often simplified nature of these models, the goal of this project was to investigate if their findings hold up when increasing the cognitive load of the pedestrian. An eye-tracking experiment was conducted where two promising eHMI signal types (i.e., flashing lights and message) were tested in a realistic scenario, where the participant took the role of pedestrian. In the experiment, the cognitive load of the participants was varied, by applying different sizes of gaze contingent windows to the stimuli.

A range of 63 different trials of 10 s each were shown to 23 participants. This set of trials included 7 different presets, aimed to test the effect of the signal type and the gaze window size independently. The participants' task during the experiment was to indicate when they deemed it safe to cross the road, by pressing the space bar. Their gaze was measured with an eye-tracker, after which it was transformed into three different metrics: saccade count, saccade amplitude and fixation duration. Additionally, a novel metric, the dispersion of the grouped gaze data, was introduced and explored. Finally, a questionnaire was conducted, investigating the self-reported clarity of the different signal types. Together with the reaction time, these metrics aimed to answer the following research question: *What influence does raising the cognitive load for crossing pedestrians have on the effectiveness of text-based eHMIs as opposed to light-based eHMIs and no eHMI?*

The results show that, as has been previously established, light- and text-based signals showed very similar response times and feature similar gaze characteristics when compared to each other. Both outperformed the condition where no signal was shown. Raising the cognitive load shows a decrease in saccade count and saccade amplitude, paired with a higher fixation duration. However, no proof could be found that raising the cognitive load has an influence on the effectiveness on any of the different signal types, meaning that either, the cognitive load was not raised by enough, or there is no actual effect. The dispersion showed that for the light-based signal, the focus on the stopping car is lost the quickest after the signal was shown.

The results of this study may help in explaining how pedestrians base their traffic decision on the type of signals being shown. This may also serve as a basis to further explore the possible effects of cognitive load on the effectiveness of these types of signals.

Keywords automated driving, external human-machine interface, gaze contingent window, cognitive load, pedestrian

Preface

With an accomplished feeling I present you my thesis on: *The effect of cognitive load on the effectiveness of external Human-Machine Interfaces.* During the time I worked on this project an eye-tracker experiment was designed, created, conducted, and analyzed. Each and every aspect of this process gave me a unique insight into the life of a researcher.

First of all, I would like to express my deepest appreciation to my supervisors, Yke Bauke Eisma and Joost de Winter, for giving me the opportunity to do this project on human-machine interfaces. It wasn't the easiest time for me when I asked them for help, yet they put their trust in my ability to make this a success. With their continued guidance I was able to bring this project to what you can see here, a finished thesis.

Second, I would like give special thanks to my thesis committee, which includes my supervisors, as well as Dimitra Dodou, for accepting and proceeding with finished thesis, and helping me with various tasks required to bring this journey to a satisfying conclusion.

Third, I would like to thank all of the respondents who participated in the experiment. Their time and best efforts were invaluable for the continuation of my project, and I really could not have done without their cooperation.

Finally, I would like to tank my family and close friends, without whom I would not have been able to stay on track. With a nudge here and there, their love and guidance kept me going and kept me believing in myself and my abilities.

I sincerely hope you will enjoy reading this thesis. I want to end with a quote by Emanuel Lasker, that stuck with me ever since I read it. He said: "Without error, there can be no brilliancy."

L. A. van Gent
Delft, July 2022

CONTENTS

1

Introduction

The developments in the field of self-driving cars are ever increasing, with more and more manufacturers and researchers around the world trying to create and implement cutting edge technology to improve and expand automation. Ultimately these technologies will replace traditional human-operated cars.

Since the Society of Automotive Engineers (SAE) has introduced their six levels of automation (SAE 2021), this standard has become a benchmark for progress in the field. The levels they defined range from 0 to 5, where level 0 describes cars that have *no driving automation*, and level 5 describes cars that have *full driving automation*. Car manufacturers are currently aiming to achieve implementation of level 3 systems for the next-generation cars, which is *conditional driving automation*. Systems such as Audi AI Traffic Jam Pilot fall under this category (Audi 2017). However, at the time of writing, systems like these are not widely implemented yet.

It is important to realize that a lot of traffic behavior is the result of human behavior. Informal interactions between humans decide for a large part how the traffic flows. This holds for car-to-car interaction, but also for car-to-pedestrian interaction. These informal ways of communication between human car drivers and pedestrians will disappear when more and more cars reach higher SAE levels. As a result, the mutual trust road users have could disappear (Creech et al. 2017). While automated cars currently have the ability to detect and act upon pedestrians, they cannot explicitly communicate their intent other than by using their lights and horn. Implicit queues related to the car, like its trajectory and its dynamics under braking, do relay some information, and is shown to be used by pedestrians to base their traffic decision on (Schmidt et al. 2019). A widely investigated traffic situation is the case of a pedestrian wanting to cross the road. Other situations have been investigated, including car-to-cyclist interactions, but for the purpose of producing comparative results, the focus of this research will be on car-to-pedestrian interaction in the aforementioned traffic situation.

Several car manufacturers and research groups have been trying to develop a framework to retain some of the car-to-pedestrian interactions with the help of an external Human-Machine Interface (eHMI) (Bazilinskyy, Dodou, and De Winter 2019, Schieben et al. 2018). An eHMI can be defined as a system that is capable of relaying information about the state or intent of the vehicle to other road users, by means of an externally mounted system. Simple forms of eHMIs are any lights and speakers mounted externally to a vehicle, meant as an extension to the car's current capability of relaying information.

Many different types of eHMIs have been explored in multiple studies and concepts. Some of the proposed ideas include mounted displays capable of showing text-, light- and symbol-based messages (Bazilinskyy, Dodou, and De Winter 2019, Clercq et al. 2019). Additional ideas that have been explored are eHMIs in the form of light-strips, showing various patterns (Böckle et al. 2017, Cefkin et al. 2019); and eHMIs in the form of projection, showing text-, light- and symbol-based messages as a projection in front of the car (Bazilinskyy, Dodou, and De Winter 2019, Suwa, Nishimura, and Sakata 2017).

From literature, two types of eHMI that are most promising in effectively relaying the car's intent to pedestrians are light-based and text-based (Clercq et al. 2019, Schieben et al. 2018, Fridman et al. 2017, Bazilinskyy, Dodou, and De Winter 2019). Thus far, text-based signals seem slightly more effective than light-based signals, the latter often being ambiguous (Bazilinskyy, Dodou, and De Winter 2019, Schieben et al. 2018).

Most of the research performed on the effectiveness of these eHMI types has been of relatively simple nature. Often, studies relied on simplified models or questionnaires only showing pictures. While their reported findings appear to be consistent, it is hard to tell if they hold when increasing the pedestrians' cognitive load. Potentially dangerous situations could occur when preoccupied pedestrians miss a critical signal.

Therefore, the focal point of this research is to investigate if the existing findings from literature hold when increasing traffic complexity in a realistic situation.

The research question we aim to answer in this research project is stated as follows:

- English: *What influence does raising the cognitive load for crossing pedestrians have on the effectiveness of text-based eHMIs as opposed to light-based eHMIs and no eHMI?*
- Dutch: *Wat voor invloed heeft verhogen van cognitieve belasting bij overstekende voetgangers op de effectiviteit van tekstuele eHMI's ten opzichte van op licht gebaseerde eHMI's en geen eHMI?*

To be able to answer the main research question, a number of hypotheses have been formulated. They are based on literature findings, intuition, and experiences with similar systems and situations:

1. H0: *Increasing the cognitive load decreases the effectiveness of the text-based eHMI*;
2. H0: *Increasing the cognitive load decreases the effectiveness of the light-based eHMI*;
3. H0: *Increasing the cognitive load decreases the effectiveness of no eHMI*;
4. H0: *Increasing the cognitive load decreases the effectiveness of the text-based eHMI more than the effectiveness of light-based eHMI*;
5. H0: *Increasing the cognitive load decreases the effectiveness of no eHMI more than the effectiveness of text-based eHMI*;
6. H0: *Increasing the cognitive load decreases the effectiveness of no eHMI more than the effectiveness of light-based eHMI.*

To be able to systematically find answers to these hypotheses, and by extension the main research question, an experiment is conducted.

For this experiment, a virtual environment is created in which several configurations of traffic can be animated. Trials are created in this virtual environment in which cars, equipped with an eHMI, drive around. When nearing the participant, one of the cars may stop to let them cross the road. When doing so, the car may show a signal on their eHMI. The way this is constructed, including certain design decisions, can be seen in Chapter 2.2.

During the trials, participants will be required to press a key when they deem it safe to cross the road, in order to measure their reaction speed.

The traffic complexity can be increased by giving participants a secondary task to perform. This means focus has to be divided between tasks, and thus increasing cognitive load. The way this is applied in this research is through the means of a gaze contingent window. More on this subject can be found in Chapter 2.3.

The EyeLink® 1000 Plus (SR Research Ltd. 2017) is used to apply this gaze contingent window during the trials, as well as to measure the participants' gaze during the trials.

From the recorded gaze data and key-presses, a qualitative investigation is performed aiming to answer the main research question. This analysis is performed in MATLAB® (The MathWorks, Inc. 2022) and an in-depth explanation of how this analysis was performed can be found in Chapter 2.5. The results of the analysis can be found in Chapter 3.

A discussion following from interpreting these results, as well as the reached conclusions answering the main research question, can be found in Chapter 4.

2 METHODS

In this chapter, the entire experiment set-up and design decisions, as well as the complete experimental procedure are walked through. Section 2.1 covers the selection of the independent variables. Section 2.2 shows how the stimuli shown during the experiment are created. The addition of the cognitive load is explained in Section 2.3. Section 2.4 elaborates how the experiment itself was conducted. Section 2.5 explains the entire analytic procedure, where the experiment results are processed and examined.

2.1. EXPERIMENT CONDITIONS

The experiment will be performed by showing stimuli on a screen, while measuring the participants' gaze with the EyeLink® (SR Research Ltd. 2017), available at the Cognitive Robotics (CoR) research group. This eye-tracker is able measure a participant's gaze at 2000 Hz with an accuracy down to 0.15° (SR Research Ltd. 2021).

In order to utilize this eye-tracker properly, the stimuli are created in-house, with the aim of looking realistic. A virtual environment representing several different traffic situations is created in which cars drive past the participant. While keeping the layout of the traffic situation the same, the order in which the cars will drive by varies. Additionally, the order in which one or more cars will stop varies.

In these stimuli, the independent variables are present. An overview of the variables can be seen in Table 2.1.

Table 2.1: Overview of the different groups and conditions in the experiment.

	Independent variable	
Condition	Signal type	Gaze window size
	FLASH	SMALL
	MESSAGE	LARGE
Control	OFF	NONE

2.1.1. SIGNAL TYPE

The first goal is to be able to compare two different signal types (i.e., flashing lights and message, with respect to a control condition). These signal types are to be shown on an eHMI, which is placed on a self-driving car. Within both of these categories of signal types, many possible implementations have been suggested and investigated before in literature.

The effect of type of text signal has been considered (Clercq et al. 2019, Barendse 2019, Song et al. 2017, Bazilinskyy, Dodou, and De Winter 2019), as well as the effect of color (Bazilinskyy, Dodou, and De Winter 2019), and the effect of positioning (Eisma et al. 2019). The main findings that are taken from these studies is that the type of text does matter. From several studies it follows that messages related to the pedestrian (egocentric), are slightly more effective than messages related to the car's intent itself (allocentric), and the car's status (affirmative) (Barendse 2019, Song et al. 2017). However, legal issues may arise when using egocentric messages (Bazilinskyy, Dodou, and De Winter 2019), so for this research, the allocentric message "Waiting", in combination with a symbol, is used, similar to signals used in previous research (Eisma et al. 2019).

Similarly, the effect of type of light signal has been considered (Clercq et al. 2019, Faas and Baumann 2019, Hensch et al. 2019, Cefkin et al. 2019, Böckle et al. 2017). Established customs in traffic interaction are also taken into consideration (Renge 2000). This results in the choice for a white flashing light signal. The eHMI turns on and off twice, signaling the pedestrian that the car is going to stop.

As a control condition, the absence of a signal is opted for. This represents the situation as it currently is. Participants can use vehicle dynamics to base their decision on.

2.1.2. GAZE WINDOW SIZE

The second goal is to compare the effect of cognitive load on these types. In order to do that, a gaze contingent window is utilized. This is added post-hoc to the stimuli (at the experiment run time). More specifically, the independent variable is the gaze window size. To identify the effect, a small and large gaze window are selected, and as a control condition there is no gaze window.

2.2. STIMULI

Blender (The Blender Foundation 2021) is used to create the stimuli.

The reason why Blender was chosen over CAD programs like ProEngineer or Solidworks is because complete physical realism is not a goal for this experiment. Complex stress and material specific calculations are not required. The goal is to create an experimental environment that is realistic and immerse for the participants. This means objects placed in the scene do not have to be physically accurate, but merely have to be accurate enough for the experimental purposes. In more concrete terms, this means the objects in the scene need to move and look realistically. This is exactly what Blender excels at, being able to render using ray tracing for example. Very realistic lighting can be achieved this way, while the materials of the objects do not have to be 100 % accurate. In the same line of thought, vehicle dynamics can be implemented in a simplified manner by using Blender's animating capabilities.

Furthermore, Blender is a very popular free-to-use application, with many online sources to cater to our needs ranging from free-to-use models and materials to tutorials and literature.

The scene that needs to be created can be split up into different groups and subgroups based on their purpose within the scene. They can all be created separately, after which they can be linked together into a single scene. In the broadest sense, we can say that we need the following basic groups: a road system with a side walk, cars with eHMIs that populate this road system, and environmental objects that make the scene look alive and obscure long-distance views.

To be able to explain the work flow in Blender, this section is split up into subsections. Section 2.2.1 will go into the static side of the scene. This includes all the objects present in the scene and how they instanced. Section 2.2.2 will show the different presets that will be created in the scene. Section 2.2.3 will describe the dynamic side of the Blender setup. This includes how the objects' movements are described and how the triggers are implemented. In short, it describes how the static scene, plus the different presets, result in the actual stimuli. Section 2.2.4 describes the rendering process and post-processing of the renders.

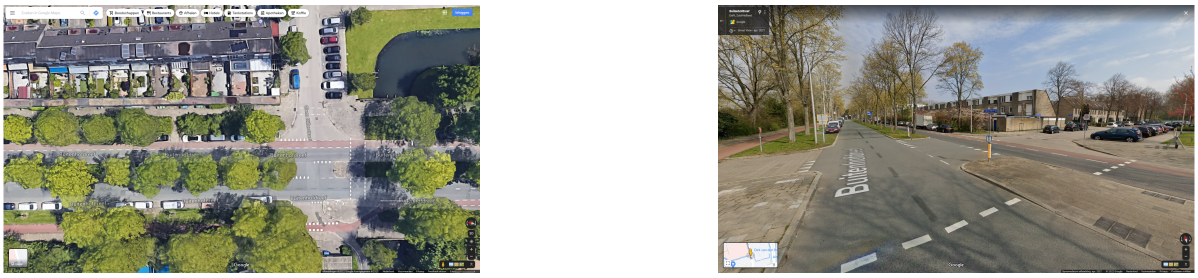
2.2.1. STATIC MODEL

This subsection will go into detail about how all the objects living in the scene are created. This varies per object, as some objects are sourced from the internet, whereas others are created in-house. When the objects are home-made, they are also textured in-house. This procedure is also described.

ROAD SYSTEM

The road system needs to meet several requirements. First of all, the system needs to be based on a real-world example, since this makes it easier to compare, should a real-life equivalent of this research be conducted. The selected intersection should be one where it is likely for a pedestrian to cross to the other side of the road, but where cars do not necessarily have to stop (but could if they wanted to). This means they should have a relatively low driving speed (e.g., a residential area). Furthermore, the cars should not be able to cross from unexpected directions from the perspective of the pedestrian. A real-life example that meets these requirements is an intersection in Delft, specifically between the Glenn Millerstraat and the Buitenhofdreef, as shown on a Google Maps image in Figure 2.1.

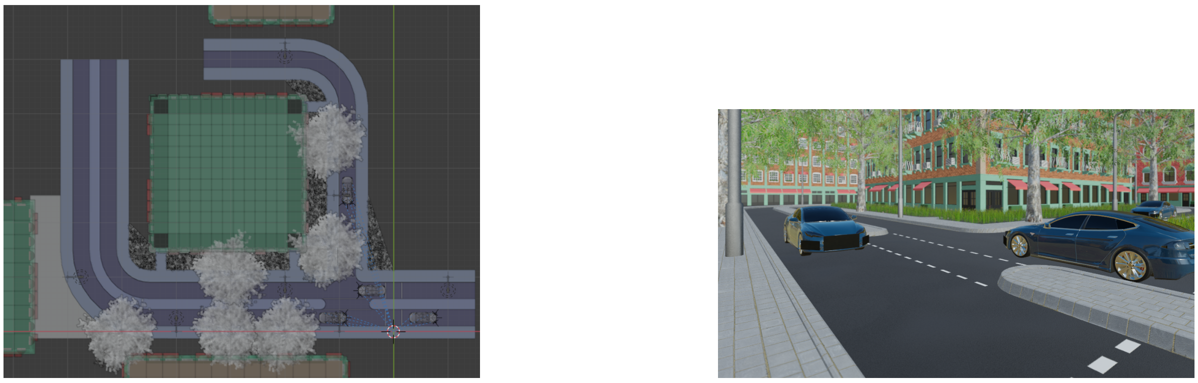
In this example, there is a two-way street people can cross with a t-crossing. The two-way street has a median, so pedestrians only have to focus on traffic from a single direction. Close to the crossing is a traffic light, so all crossing cars are either slowing down, or turning into the road. Some simplifications were made to make the situation more controlled. For example, all the roads are turned into single lane one-way roads, so cars cannot overtake, and multiple cars cannot cross from the same direction at the same time. This leaves us with 3 possible directions for the cars to drive in. All these 3 directions are turned into splines in Blender.


(a) Top-down view of the intersection (Google, 2022)

(b) Street view of the intersection (Google, 2021)

Figure 2.1: Intersection of the Glenn Millerstraat with the Buitenhofdreef (Google Maps).

To create these splines in Blender, the Google Maps image of the intersection is used as a reference. The splines can then be traced on top of the reference image. From these splines, the actual road model is created, based on real-life standard road values (e.g., a set width). Afterwards, the side-walks and the median are added to this model. The scene is populated further with props to make it look more realistic. The final result, in the form of a top-down view, and the pedestrian point-of-view, can be seen in Figure 2.2.


(a) Top-down view of the intersection (Blender Viewport Render – Solid Mode)

(b) Pedestrian point of view of the intersection (Blender Viewport Render – Material Preview Mode).

Figure 2.2: Resulting Blender model including all props, displayed using Blender's Viewport Renderer.

The splines are also used later on, when animating the cars on top of them, so they have to be detailed enough. In a vertex-based program like Blender a circle is never really a circle. All vertices are joined by straight edges. Depending on the number of vertices in the spline, the edges form the appearance of a circle (the higher the number of vertices, the more the spline approaches a real circle). The cars following the splines need to turn realistically. When the spline's resolution is too low, the cars' turns will look jagged.

Road textures The asphalt texture is procedurally generated to avoid creating a tiled look. Procedural generation for textures in Blender is easy to set up using the node-based Shader interface. Using the road object's coordinates as an input for a structured noise pattern, a grain-like surface can be created that is not repeating. Material properties of this grain-like surface can be modified by tweaking several Shader nodes based on this noisy input.

For the side walk the opposite is needed. A material created with physically based rendering (PBR) can be used to create a tiled look. Materials with PBR maps can be applied in Blender in a way that gives the surface photo-realistic detail, including apparent displacements and material-specific reflections. This makes them very suitable for the purpose of this experiment. A material with high resolution PBR map licensed under the Creative Commons CC0 license is used for the side walk (ambientCG 2019b).

Lastly, road markings are required to make the road look like a real road. To do this, a layer is created slightly above the road. A procedurally generated paint-like material is applied using the node-based Shader interface. Using the alpha-channel, the sides of the paint can be made slightly erratic, making it look like someone actually painted on the road.

CAR MODEL

The car model was sourced from the internet. A modern looking car is preferred as people associate these with having the ability to be self-driving. A model of the Tesla Model S was used for this scene (C.C. Studio 2019).

Editing the model To make the model suitable for the scene, editing was required to make it appear functioning.

In order for the wheels to be able to rotate as the vehicle moves, the wheels are separated from the source model and re-added as separate objects. In Blender terms, they are parented to the car origin, meaning the wheels' local axes are relative to the car's local axes. This, in turn, means that the position and orientation of the wheels will follow that of the car's origin. For the rotation and the orientation, the variables from the Animation Nodes set-up were used (see Section 2.2.3 for more information). Using the traveled distance plus the wheel offset, the exact position and orientation of each of the wheels is calculated by taking the tangent of the spline with respect to the global axes at every position. The car and the rear wheels have the same orientation. The front wheels are used for turning and can therefore have a different orientation. The traveled distance is also used to calculate the phase of the wheels, so they rotate realistically when advancing along the spline.

eHMI For the actual experiment, the cars need to have an eHMI that triggers when the car stops. The eHMI in the experiment has two modes that can both be turned on and off: light-based and text-based. This requirement is realized by implementing this as a conditional set-up. A trigger is used to activate the eHMI once the car has traveled a certain amount, and meets the other activation requirements.

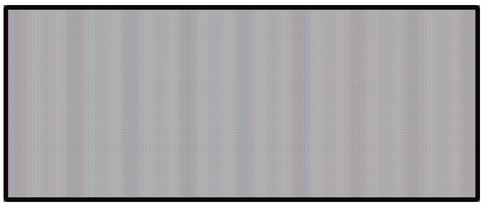
The eHMI panels used in the scene were designed from scratch. To make the LCD-panels as realistic as possible, they were set-up to have a specific size: $1000 \times 400\,\mathrm{mm}^2$, such that a $1000 \times 400\,\mathrm{px}^2$ image could fit in exactly. Around this panel a case was constructed. The case was given a simple plastic material, while the panel was given a special material set-up.

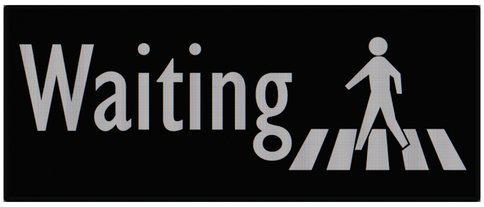
In particular, the material consists of a couple important node groups. First of all, when looking at a single pixel from a real LCD-screen, it can be seen that it actually exists of three separate lights (red, green and blue) that can be turned on or off based on a computer signal. To replicate this look, a node group was created to fit the $1000 \times 400\,\mathrm{mm}^2$ panel with 1000×400 square RGB-pixels. If then mixed using a Mix node with a real image of the same size, these simulated pixels will behave exactly like a real LCD-pixel. Secondly, multiple inputs were made to simulate a screen that is either: off; displaying a text message; or displaying a light message. Then, finally, different Mix nodes are used be able to switch between these different inputs, controlled by triggers in the Animation Nodes set-up.

Three images are created to be displayed, following Table 2.1 and can be seen in Figure 2.3. The first image is completely white. This image is used when the screen is displaying the flashing lights signal type. The second image is one that displays text alongside a crossing symbol, used for the message signal type. The symbol is taken from the design used on road signs in The Netherlands (Ministerie van Verkeer en Waterstaat 1990). The font of the text was chosen such that the rest of the available area of the display is optimally used. The third image is completely black. This image is used when the screen is not displaying any signal (off).

The same color was chosen for both signal types. This is a deliberate choice. From the literature it follows that it is best to use neutral colors to display a message, to take away potential ambiguity (Bazilinskyy, Dodou, and De Winter 2019).

eHMI position on the car A suitable position for the eHMI panels is selected. From literature it follows that people tend to look at multiple aspects of an approaching car (Eisma et al. 2019). First of all, they try to


(a) eHMI panel displaying the flashing lights signal type.



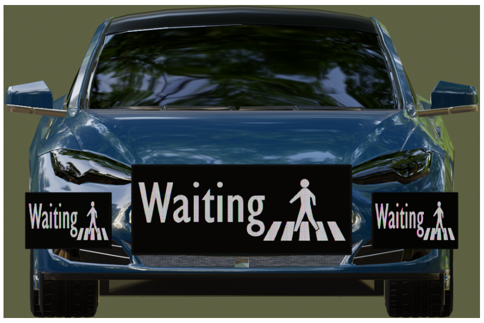
(b) eHMI panel displaying the message signal type.

(c) eHMI panel displaying nothing (off).

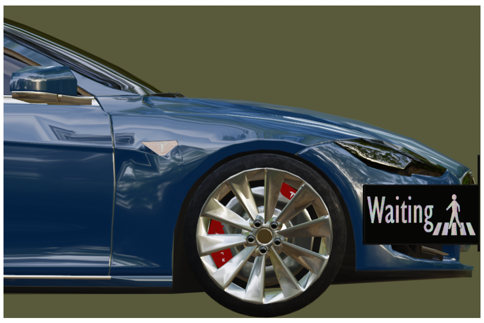
Figure 2.3: The three different signal types displayed on the eHMI panel.

observe the driver, to see if they have been seen. Secondly, they look at the front of the car and the close to the wheels to observe the vehicle dynamics in order to see if the car slows down. To be able to observe the signal at multiple angles, the eHMI is placed on the front of the car, as well as closer to the wheels. This results in a triple panel setup shown in Figure 2.4.

This way, the panels are visible when the car is approaching head on, but also when turning into the road. The low placement of the panels is in line with the expectation that the participants are going to look at the bumper and close to the wheels. Note that the wind shield is not see-through so the driver cannot be observed.



(a) Front view showing the positioning of the eHMI panels on the car model.



(b) Side view showing the positioning of the eHMI panels on the car model.

Figure 2.4: Front and side view of the car model, showing the positioning of the eHMI panels.

ENVIRONMENT

Alongside the road, buildings are placed to conveniently obstruct vision, such that the cars and their paths can be set up, and an entire vista would not have to be constructed. This way, the cars can spawn in obstructed vision, and anything behind it would not need to be rendered.

Buildings The buildings used in the scene are part of a collection used to showcase the Blender Geometry Nodes (Del Rio 2021). The buildings look clean enough to use as distant vision-blocking objects, and the good thing is that with the Geometry Node set-up, they are scalable to fit the scene's needs (i.e., stretch them far and high enough to completely block unwanted vision).

Lamp posts A lamp post was created to represent a Dutch looking one. The applied material mimics the look of galvanized steel. The lamp itself consists of 2 fluorescent tubes, encapsulated in plastic. However, since it is a daylight scene, the actual lights are turned off. The posts are imported into the scene and spread at regular intervals along the road using the Geometry Nodes Blender utility, which instances the objects using a node setup.

Trees The trees used in the scene are part of a collection sourced from the internet (Cardona 2021). The tree used is a single one from the collection, rotated along its z-axis randomly and placed at regular intervals along the road using Geometry Nodes.

Grass The grassy fields consist of a collection of grass, flowers and rocks, sourced from the internet (Cardona 2021), which are spread around predesignated areas in a semi-random way using Geometry Nodes. The ground itself is a material with high resolution PBR map licensed under the Creative Commons CC0 license (ambientCG 2019a).

HDRI To make the environment even more realistic, a 360° high dynamic range image (HDRI) is used to wrap around the entire scene (Majboroda 2019). This gives the illusion of the scene being in an actual environment. Blender is also able to use this image as a lighting source for the entire scene, meaning the sun in the image will be the actual source of light in the scene. This will result in logically positioned shadows and reflections, adding to the immersion.

2.2.2. PRESETS

When it comes to the experiment itself, several different scenarios within the same scene will be shown to the participants. Each scenario is a preset described by several values used in a Python script, which calculates the cars' positions over time (see Section 2.2.3 for more information). Depending on the preset, the cars can follow one of three splines until they either stop in front of or drive past the participant.

The splines are labeled and numbered as routes in order to describe the presets. The preset order is defined as the order the cars will pass the participant in. The cars that will stop will be marked as such.

A visual representation of the routes can be seen in Figure 2.5.

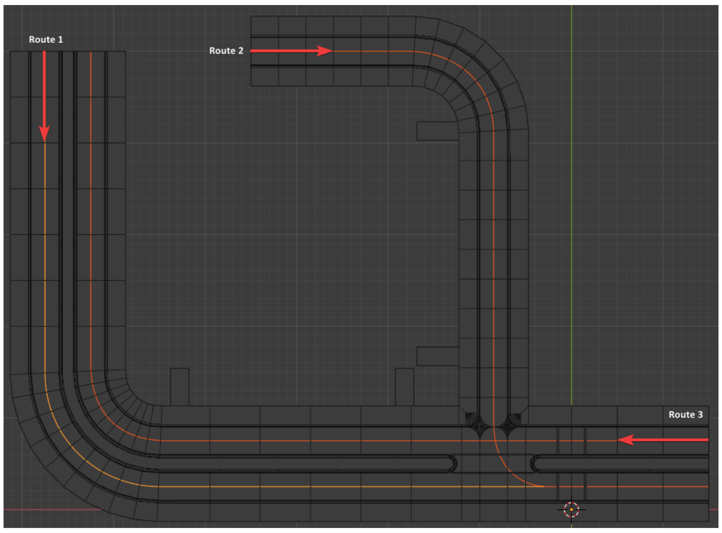


Figure 2.5: Top-down view of the three different routes in the scene.

With this map in mind, seven different presets were created. The presets were chosen in such a way that the otherwise similar traffic situations are diverse. The cars can drive on all the routes, but do not always stop. An overview of all the different presets is presented in Table 2.2.

Since it is already known that only the signal type independent variable needs to be added during the construction of the stimuli (the gaze window size will be added post-hoc – recall Table 2.1), three different stimuli need to be created for each of the seven presets. This results in a total of 21 stimuli.

Table 2.2: Mapping of the different routes and orders between all trials, including the stopping conditions of all cars.

	Preset 1		Preset 2		Preset 3		Preset 4		Preset 5		Preset 6		Preset 7	
Car	Route	Stop	Route	Stop	Route	Stop	Route	Stop	Route	Stop	Route	Stop	Route	Stop
1	1	no	3	no	1	no	2	no	2	no	3	no	3	no
2	2	no	1	no	1	no	1	no	1	no	2	no	2	no
3	1	no	2	no	2	yes	2	no	2	yes	1	yes	1	no
4	1	yes	1	yes	3	no	1	yes	1	yes	1	yes	1	no

2.2.3. DYNAMIC MODEL

This subsection will go into detail on how the static Blender objects are linked together, and move around the scene. This includes how the python script, controlling the initial conditions, is set up. Subsequently, this also includes how the node-based Animation Nodes controls the positioning of the objects throughout the scene for each frame.

DISTANCE PROFILES

To be able to be in full control over Blender's graphical abilities, scripts can be created to precisely turn the dials. This also overcomes several shortcomings in Blender's user interface.

Since Blender features the ability to edit an object's world position over time, there are several methods to simulate an object's velocity and acceleration over time as well. The usual way to animate an object is to set the object's location at two different points in time, and let Blender's internal scripts interpolate a path between these two points. The interpolation method can then be tweaked further to realize an approximation of what is required.

However, if the acceleration, velocity and traveled distance of each car are known at every point in time, there is no need for an approximation. For this purpose, Blender's default animation method is not used, and instead, a script is made to pre-calculate the trajectories instead, creating distance profiles for each of the cars. Additionally, checks for the stopping conditions and the eHMI signals are included.

Blender has built-in Python integration which lets us read and write object properties through the means of the Blender Python module (`bpy`).

Blender features the option of running a script either on frame basis, or once upon loading the scene. Running a script every frame allows for adaptive behavior, letting objects react to events without explicitly knowing when they will happen. However, since all the cars' trajectories can be determined beforehand, we can opt for the faster option, which is executing a script only once upon loading the scene. This results in all the objects of interest to be instantiated before running the animation.

In Blender, every object in the scene is present in the same format. The name of the object can be used to select the object and read or alter its properties. From the world scene the car objects will be selected, and each of them will be assigned to a certain trajectory, according to Table 2.2. After calculating the trajectories for each of the cars, these will be written to each Blender object as custom properties. Afterwards, these custom properties can be used by the Animation Nodes add-on within Blender to animate the objects' trajectories along the assigned routes over time.

The complete Python script utilized by Blender can be seen in Appendix C.2. In the script, each car object is initiated as a Python class object. Using a class method, the following equations were used to calculate the acceleration a, velocity v and traveled distance s:

$$s_{t_{n+1}} = s_{t_n} + v_{t_n}\Delta t + \frac{a_{t_n}}{2}\Delta t^2, \tag{2.1}$$

$$v_{t_{n+1}} = v_{t_n} + a_{t_n}\Delta t, \tag{2.2}$$

$$a_{t_{n+1}} = a_{t_n}. \tag{2.3}$$

In these equations a, v and s are functions of time t. In Blender, frames are used to indicate the time. n represents the current frame number such that t_n represents the time t at frame n. Additionally, Δt is the time between each two consecutive frames. We iterate over these equations starting from $n = 1$, at which $t_n = t_1 = 0$, until $n = n_{\text{end}}$. The length of the animation is determined by the combination of the total amount of frames n_{end} and the frame rate r. After all,

$$t_{\text{end}} = \frac{n_{\text{end}}}{r}. \tag{2.4}$$

After setting a frame rate, Δt can be determined as follows:

$$\Delta t = \frac{1}{r}. \tag{2.5}$$

The cars have four states they can be in: standstill, accelerating, cruising, decelerating.

Standstill (start) All cars start in the standstill state. In this state, the car's position is at a certain distance $s_0 = s_{\text{traj},0}$ from the pedestrian (note that $s_{\text{traj},0}$ is different for each trajectory).

In the calculations, traveled distance s is negative until it reaches the pedestrian at $s = s_{\text{p}} = 0$, and is positive after it has passed the pedestrian.

Accelerating When a car needs to start driving, its state changes to accelerating. In this state, the car starts with a velocity $v_0 = 0\,\text{m}\,\text{s}^{-1}$ and a constant acceleration of $a_0 = 5\,\text{m}\,\text{s}^{-2}$. During this phase, the car increases v linearly until target velocity $v_{\text{T}} = 8\frac{1}{3}\,\text{m}\,\text{s}^{-1}$ $(= 30\,\text{km}\,\text{h}^{-1})$ is reached.

Cruising The car will enter the cruising state when the calculated v_t satisfies the following equation:

$$v_t \geq v_{\text{T}}. \tag{2.6}$$

In this state $v_t = v_{\text{T}}$ and $a_t = 0$. This means acceleration is stopped and a constant speed of $v = v_T$ is set. During this phase, the car keeps driving at the constant velocity $v = v_{\text{T}}$. If the car is meant to stop, it will continue driving at this velocity until $s_t = s_{\text{T},k} - s_{\text{b}}$.

Decelerating The car will enter the decelerating state when the calculated s_t satisfies the following equation:

$$s_t \geq s_{\text{T},k} - s_{\text{b}}. \tag{2.7}$$

In this state $a_t = a_{\text{d}}$. This means the car decelerates linearly until $v_t = 0$.

This decelerating state will only be entered when a car needs to stop for the pedestrian. When this is the case, the car does so at $s_{\text{T},k} = -5k - 2(k-1)\text{m}$, where k is the kth stopping car. This means that the first stopping car will do so at $s_{\text{T},1} = -5\,\text{m}$, and every subsequent car will stop 5 m behind the previous one (taking into account the car length of $\sim 2\,\text{m}$). This, in turn, means $v_{\text{end}} = 0\,\text{m}\,\text{s}^{-1}$ and $a_{\text{end}} = 0\,\text{m}\,\text{s}^{-2}$ at $s_{\text{end}} = -5\,\text{m}$. To achieve this, it will have to start decelerating a little bit before it reaches that point. A deceleration called a_{d} is introduced and described as follows:

$$a_{\text{d}} = -a_0. \tag{2.8}$$

The braking distance s_{b} then becomes:

$$s_{\text{b}} = -\frac{v_{\text{T}}^2}{2a_{\text{d}}}. \tag{2.9}$$

Note that for the cars that are not meant stop, the constant velocity $v = v_{\text{T}}$ is maintained until $n = n_{\text{end}}$, which marks the end of the simulation.

Standstill (end) The car will enter the standstill state again when the calculated v_t satisfies the following equation:

$$v_t \leq 0. \tag{2.10}$$

In this state $v_t = 0$ and $a_t = 0$.

The resulting acceleration, velocity and traveled distance profiles for a set of objects is shown in Figure 2.6. The cars have an offset in starting time, which was tweaked manually to give the cars a natural looking following distance and merge timing. The car that is meant to stop does so at the predefined stopping distance, while the other cars continue driving at a constant velocity. The marked key frames can be seen in Figure 2.7.

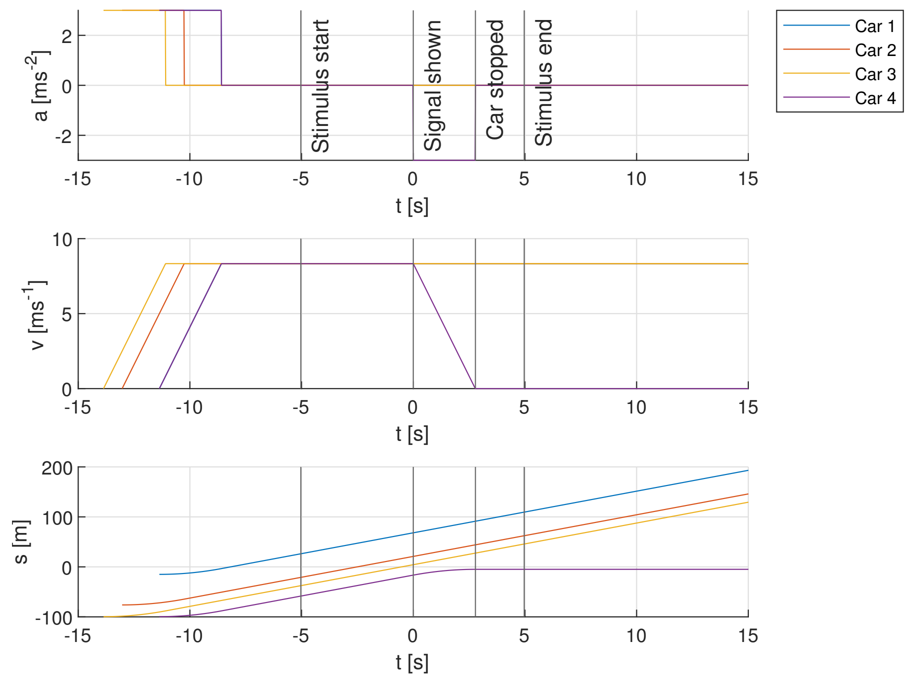


Figure 2.6: Calculated acceleration (a), velocity (v), and traveled distance (s) profiles per `Car` object. The profiles shown are used in preset 1. The traveled distances are normalized around the participant's position ($s > 0$ indicates a car has passed the participant). The complete set of profiles can be seen in Appendix A.

ANIMATION

To animate the positioning of the cars over time, the distance profiles resulting from the Python script are applied to the scene with the use of the Animation Nodes add-on (Lucke and Emara 2019) in Blender.

The Animation Nodes add-on allows for a versatile animating set-up, while at the same time being intuitive and in line with several other node-based environments within Blender like the Shader and Geometry Nodes, used in Section 2.2.1.

The node set-up created in this add-on is executed every time a new frame is selected in Blender. This allows blender to go through the list of positional values calculated by the Python script frame-by-frame, placing the cars at the right position of their predefined routes for each frame.

More specifically, each car's traveled distance for each specific frame is matched to a position on the matching spline. This position, including the orientation of the spline with respect to the z-axis at this position, is then evaluated against the world coordinates after which a transformation matrix is generated that updates each car's position and orientation in the world frame.

The car's front wheel rotation around z-axis is calculated using the same method, but using an offset from the car's origin. This means the wheels orientation will follow the spline perfectly ahead of the car.

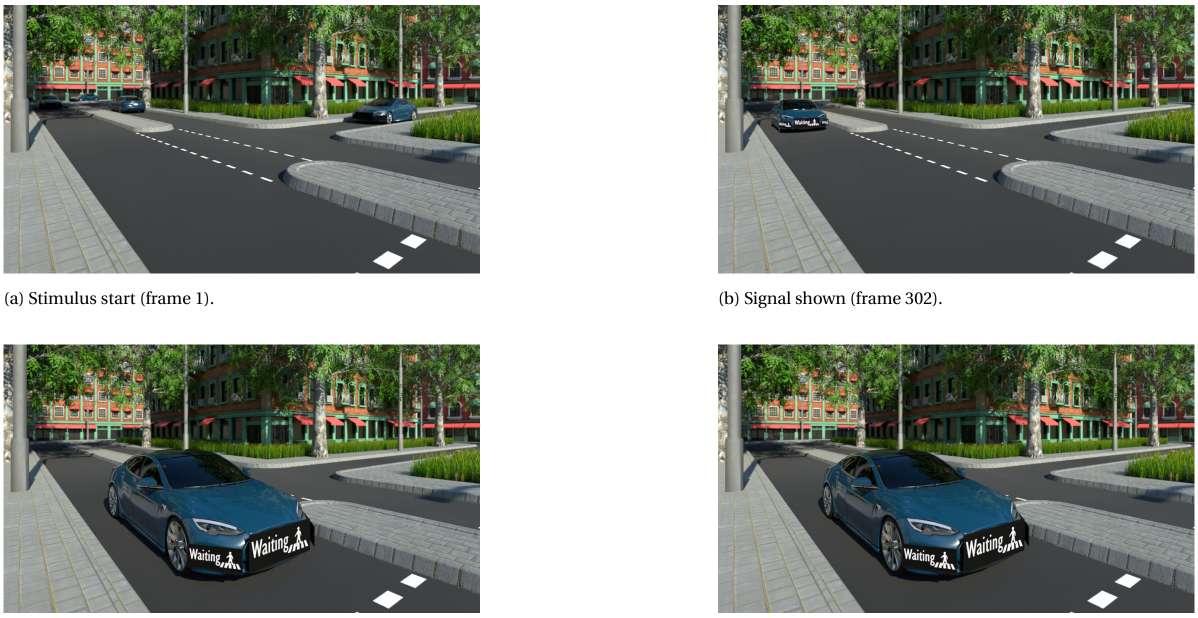
The rotation of the wheels around its local y-axis while driving is also controlled in this loop. Here we can decouple the breaks from the actual wheels, so they do not rotate along with the wheels while driving. The car's traveled distance is used to generate a phase offset in the wheels, which updates along with the cars traveled distance, effectively rotating the wheels around its y-axis.

Finally, the eHMI is controlled in this section as well. The signal type is set in the Python script, as well as on which car it is activated. This information is read by the node set-up and applied internally to that car object's eHMI panel Shader.

2.2.4. RENDERING

Rendering the scene in Blender is not a trivial matter. All frames for every scene have to be rendered individually. Furthermore, there are many settings and options affecting the quality of the render. The most important settings are investigated and the values that were chosen are explained here.

To start things off, there are many different render engines Blender can use. Some are shipped with the program; others are separate projects that can be imported. The shipped engines are Workbench, Eevee (The



(a) Stimulus start (frame 1).

(b) Signal shown (frame 302).

(c) Car stopped (frame 469).

(d) Stimulus end (frame 600).

Figure 2.7: Key frames of preset 1, with one of the eHMI types at set frames. Key frames for the complete set of presets can be seen Appendix A.

Blender Foundation 2018) and Cycles (The Blender Foundation 2019). Other engines that can be used by Blender, but have to be installed separately include the open-source LuxCoreRender (LuxCoreRender project 2017), AMD Radeon® ProRender (AMD 2017), OTOY OctaneRender (OTOY 2012), and the open-source appleseed (The appleseedhq Organization 2010).

The most basic render engine that comes with Blender is Workbench. This engine renders the 3D viewport in real-time with the same options as if working from the 3D viewport. It is not meant for creating final renders.

The Eevee render engine (The Blender Foundation 2018) creates the image by using a method called rasterization. This method works by projecting all the scene objects onto the camera surface. Then, the pixels' colors are tweaked by their object's shaders, normal and other preset properties, like whether or not an object is in a shadow. This method is much faster than the Cycles render engine, but has many limitations for which the engine uses tricks to make up for (e.g., reflections, refractions, shadows). This can result in inaccurate results, especially for animations, where certain properties are not always updated for every frame.

The Cycles render engine (The Blender Foundation 2019) creates the image by using a method called path tracing. Technically speaking, this method integrates over all the luminance arriving at a predefined point (i.e., a camera pixel). In practice this works by casting rays from each pixel of the camera into the scene, which bounce off objects until they hit a source of light, or reach the preset bounce limit (resulting in pure black for those specific traces). The algorithm casts many samples per pixel with a random offset, to include sub-pixel differences, and averages the result. The higher the number of samples, the less this method is prone to noise. Furthermore, this engine uses computational restrictions to minimize rendering time. In practice, this will only result in noticeable effects in some very specific scenarios where light calculations are complex (e.g., caustics, dispersion). For this reason this render engine is categorized as a biased.

The open-source LuxCoreRender (LuxCoreRender project 2017), known earlier as LuxRender, is a path tracing render engine like Cycles. But unlike Cycles, LuxCoreRender is actually unbiased, meaning its render result will always converge to something physically accurate. Where Cycles path tracing method only shoots rays from the camera, LuxCoreRender combines this with information obtained from rays shot from light sources. In recent versions, it includes methods to speed up rendering without sacrificing any accuracy. This makes it a very good alternative to Cycles. LuxCoreRender is able to work with Cycles materials, but for the best render results it requires engine specific materials.

Radeon® ProRender (AMD 2017) is a relatively new render engine. It is an engine that is very attractive to the MacOSX scene as it fully supports Apple's Metal graphics library, unlike Cycles. It lacks features compared to Cycles or LuxCoreRender, and its results therefore might not be as accurate. Furthermore, Cycles materials need to be converted before they can be used by Radeon® ProRender.

OctaneRender (OTOY 2012) is an unbiased GPU-only render engine. This product is aimed for commercial use, but has a free version that is limited to a single GPU set-up. In order to use OctaneRender a custom Blender build will need to be installed, and therefore will not be an ideal option as compatibility with other plugins therefore cannot be guaranteed. Furthermore, OctaneRender is not compatible with Cycles materials.

appleseed (The appleseedhq Organization 2010) is an open-source CPU-only render engine. Even though appleseed is very competent in the accuracy of its render results, without the ability to utilize the GPU, this engine will not perform better than Cycles or LuxCoreRender in terms of render time. Furthermore, appleseed is not compatible with Cycles materials.

Other render engines that need mention are YafaRay (YafaRay Team 2016) and RenderMan (Pixar 2015). YafaRay (YafaRay Team 2016) is an open-source project that compares to LuxCoreRender and appleseed in terms of render options, but is relatively obscure and does not seem to have a great community. RenderMan (Pixar 2015) is Pixar's render engine that previously only worked with older versions of Blender but recently has been announced to be updated to work with newer versions of Blender as well. However, as of writing it does not seem to be the case yet.

When taking all the available engines into account, it makes sense to first experiment with the engines that are shipped with Blender. Eevee excels when its result approximates Cycles', in which case it is much cheaper to use. However, these seem to be fringe cases and since the aim for the experiment is apparent realism, as opposed to true realism, Cycles will be the preferred render engine. After Cycles shows promising results, LuxCoreRender could be installed to see if it can improve the render-time, without sacrificing quality. The tricks Cycles uses to minimize its render-time might outweigh the tricks LuxCoreRender uses, without any noticeable differences, but without testing, this is hard to say.

Render Settings

Obtaining the right render settings is an iterative process. The aim is to render the stimuli at a resolution of 1080p. All rendering is performed on a computer with the following specs: GIGABYTE™ Z390 GAMING X Motherboard, Intel® Core™ i5-9600KF CPU @ 3.70 GHz, NVIDIA® GeForce™ GTX 1660 SUPER GPU, using 2×16 GB physical memory (CORSAIR™ VENGEANCE® LPX CMK32GX4M2D3600C1).

With the default settings as a starting point, several settings are tweaked in order to optimize the render time, without noticeably affecting the render quality.

The default number of samples per pixel used by Cycles is 128. The higher the number of samples taken, the better the end result will look. It is pointless to take a sample size that is too high as the trade-off is very noticeable in render time, as the render time scales almost linearly with the number of samples. A sample size that is too low however, will result in a very noisy image.

Denoising can be used to attempt to remove noise caused by a low sample count. Noise can be especially noticeable on single-color surfaces. The denoiser will smooth the result, effectively removing the noise by adding blur, potentially resulting in a visibly less accurate resulting render. It is key to find a high enough sample count such that the denoiser will not visibly affect the result and at the same time, not inflate the render time.

After experimenting with a couple of sample counts, 64 samples per pixel was opted for. With this setting, the denoiser applied in post-processing was able to virtually eradicate all the noise introduced by the lack of sampling.

An experimental Cycles setting that can be used to improve render time is the adaptive subdivision setting. Subdivision is necessary to make the mesh look smooth. If a mesh is not smooth, it will be very noticeable when an object is near the camera. For far-away objects however, it is not necessary to have a high subdivision setting. The higher the subdivision setting is, the longer the render-time. The adaptive subdivision setting ensures that nearby objects have a high enough subdivision setting, while objects that are further away, use a lower setting, saving render-time.

Caustics can be turned off. This setting is meant for optic refractions, which is a very resource intensive effect. Since the scene does not have any close-ups of glass or other see-through materials, it is unnecessary to take this optical effect into account.

Initially, with all these settings tweaked, it still takes up a tremendous amount of memory to build and render the scene, often causing the computer to completely fill up all available RAM and vRAM, resulting in a system crash. The only way to reduce memory is to reduce the number of vertices present in the scene.

Since the car model used is incredibly detailed, some simplifications were introduced, by editing the level of detail in the car. The vertex count of original car model equals $\sim 1.6 \times 10^6$. After carefully applying several modifiers for each surface, the vertex count is reduced to $\sim 3.3 \times 10^5$, five times as small as the original. After this modification, the computer uses significantly fewer resources, and does not crash while rendering every other frame.

Blender has the option to either output the render as an image file (single frame render), or as a movie file (sequence render). When outputting directly to a video file, any crash will render the end result corrupted. To avoid this, rendering is done per frame to separate files. After rendering each frame separately, the resulting frames are sequenced together into a single video file in post-processing. This has the additional benefit of being able to render frames in batches.

The file format chosen for the output is Multi-layer OpenEXR (EXR). This allows for saving all render passes separately, to be able to use them afterwards in post-processing. In practice the only extra data we want to save is the noise data, such that denoising can be applied and controlled afterwards. If one would save all possible passes into the EXR file, the file size per rendered frame equals ~150 MiB. With only the beauty pass (the renders' final output) and the noise pass (all the noise data to be used by the post-hoc denoiser), the file size equals ~50 MiB per render. At 60 fps, this adds up to ~28 GiB of rendered frames for each 10 s video clip.

Since three versions of every scenario need to be created (one for every signal type), the pre-signal phase only has to be rendered once. Once the signal is being shown and the vehicle is stopping for the pedestrian, the three versions differ. This saves a lot of rendering time and storage space.

POST-PROCESSING

After rendering all frames into individual files, these files are loaded in a new Blender file used for post-processing exclusively. In this file, the only environment that is used is the Compositor. It loads in all the EXR files as a video strip, after which they are fed through a node-based set-up where each of the renders is processed one after another. After processing, the composited output is encoded to a video file used for easy playback.

The task of the compositor is to denoise the image, and color correct it. Denoising is performed using the Denoise Node. This applies Intel® Open Image Denoise (Intel® 2018), which uses pre-filtered albedo and normal buffers included in the denoise pass from the initial render output. Combined with the noisy image, this denoiser is able to smooth out all the noise without leaving artifacts in our image.

Because the resulting image is too smooth, A small percentage of the noise is mixed back into the image, giving the output a little bit of grain.

Finally, a small color correction is applied to make the final image look a little warmer.

A comparison of the rendered image (beauty pass) and the final image (with denoising and composition filters applied) can be seen in Figure 2.8.

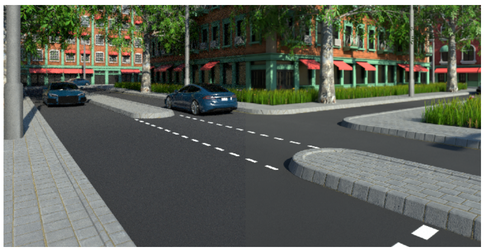


Figure 2.8: Denoising the rendered image. The left side of the image shows the noisy render output, the right side shows the same output with the desoiser applied.

Encoding The result is encoded to the widely used Matroska Video (MKV) container, able to be played back by all standard media players. The codec used for the video is H.264 with the near-lossless preset. This preserves as much detail as possible, without needlessly inflating the video size. Using these settings, each of the 21 stimuli (10 s, 1080p at 60 fps) result in a ~ 13.5 MiB file.

2.3. COGNITIVE LOAD

To quantifiably increase the cognitive load, a gaze contingent window is used. In a gaze contingent window, a tunnel-like window is created that sticks to your gaze. Different types of windows have been used in literature, for example ones where the point of focus gets amplified (zoomed), obstructed (the point of focus is obscured), or inversely obstructed (everything around the point of focus is obscured).

The point of applying a gaze contingent window is to force the participants' focus on the screen to a narrow subsection of the screen: the window. The idea is that this requires the participant to scan around the area for information more, artificially increasing the cognitive load.

The reason why blur is chosen instead of blacking out everything around the point of focus is because the participant should always be able to notice the signal, when it is presented to them. When it is visible from either the point of focus, or from peripheral vision, this should prompt an acute response, rather than a delayed or even absent response from the participant.

A gaze contingent window is easily applied with the SR Research EyeLink® eye-tracker, as it can use the output of the gaze measurement as an input to the displayed frame, in real time. Scripting is used to apply a custom window shape (SR Research Ltd. 2020a).

For the purpose of this research, a window where everything around the point of focus is heavily blurred is opted for. The aim is to create something similar to foveated imaging, which is used in virtual reality applications.

The EyeLink Experiment Builder needs two sets of stimuli files, plus a mask file to apply the gaze window. The first set contains the original, clear stimuli. The second contains the blurred stimuli, used for parts of the stimulus that lay outside of the gaze contingent window. The mask file is used to apply the shape of the gaze contingent window, in this case a circle. A schematic representation of how this works in practice is shown in Figure 2.9.

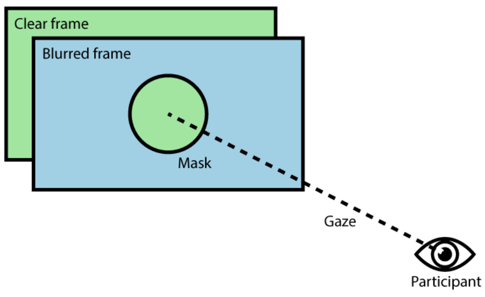


Figure 2.9: Schematic representation of the way the stimulus is displayed when applying the gaze contingent window. The blurred stimulus is on top of the clear stimulus, and the mask creates a window through the blurred stimulus at the point of the participant's gaze position, showing the clear stimulus within this window.

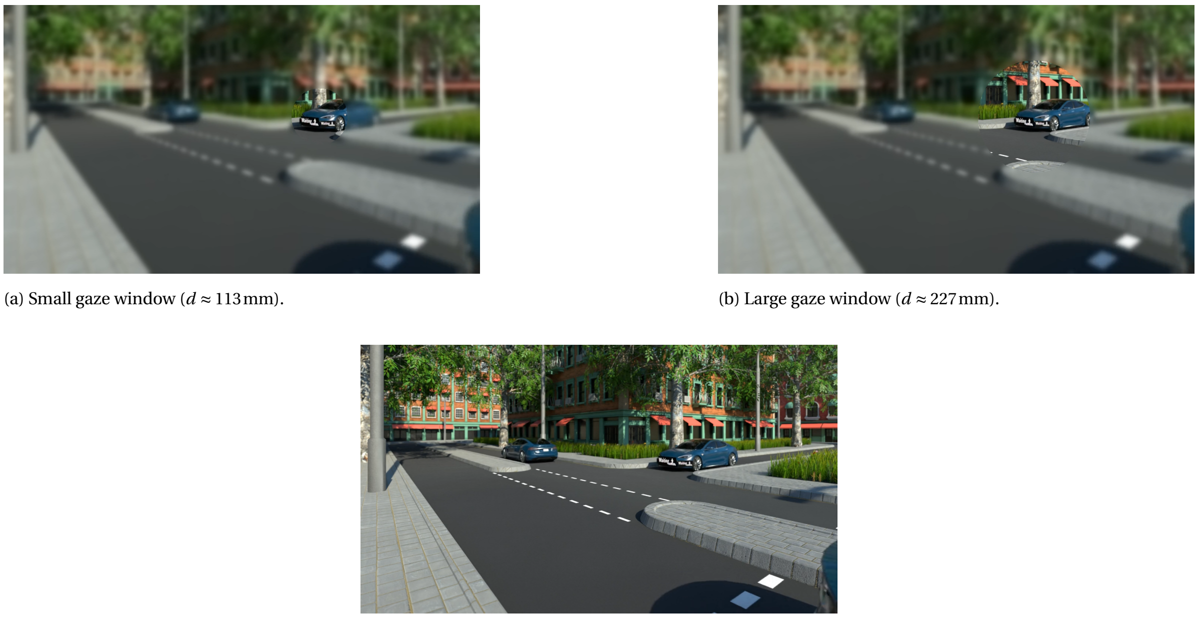
Encoding FFmpeg (The FFmpeg Developers 2021) is used to apply Gaussian blur with $\sigma = 10$ to the original stimuli. The number of steps used for Gaussian approximation is 6. This creates a relatively strong blur in contrast to the original, clear stimulus. The code can be seen in Appendix C.3.

Varying the gaze window size lets us measure the effect of the gaze window size on the participants' gaze characteristics, as well as the response time. A small, and a large gaze window size were selected, with no gaze window as a control group (recall Table 2.1). The result of the different gaze window sizes applied with the Gaussian blur can be seen in Figure 2.10.

2.3.1. OTHER CONSIDERATIONS

Before the gaze contingent window was selected as the way to vary the cognitive load in this experiment, other options were also considered:

- Changing the situation complexity:
  - Use different traffic situations;
  - Use different number of visible cars in the situation;
  - Add different props to the scene (making the scene busier).


(a) Small gaze window ($d \approx 113\,\text{mm}$).

(b) Large gaze window ($d \approx 227\,\text{mm}$).

(c) No gaze window.

Figure 2.10: Example of the gaze windows used in the experiment. The three gaze sizes (small, large, none) are applied to preset 3 with the message eHMI type at frame 302 (signal shown) for comparison.

- Adding a secondary task:
  - Memorization;
  - Same-different (symbolic or non-symbolic);
  - Auditory;
  - Counting (forwards or backwards).

The first set of options includes ideas that can increase the cognitive load through the increase of the situation complexity. While this sounds promising, there are various issues associated with this method. It is hard to quantify how much a change in the complexity actually attributes to an increase in cognitive load. Furthermore, it increases the time required to build the trials, since only part of the scenes can be reused, when the layout, number of props, or number of visible cars changes between trials.

The second set of options includes ideas that can increase the cognitive load through the addition of a secondary task.

A memorization task is one of those (Goh, Pearce, and Vas 2021). An alphanumerical sequence could be shown before each trial, after which the participant performs the trial as normal. After completing the main task, the participant would be asked to recall the initial sequence. A task like this is not expected to affect the gaze data. A downside to this method is that the performance of this secondary task would have to be verified for each instance, and when wrong, the trial results would need to be binned.

Similar to a memorization task, a symbolic same-different task could be used. In this type of task, the participant is shown two symbols during the experiment in a predesignated area on the screen, and is required to identify whether or not they are matching. Alternatively, a non-symbolic same-different task (using alphanumerical characters) could also be used. Using this type of task could result in a lot of the recorded gaze data being focused on the secondary task. This may be an indication of cognitive load and does not necessarily have to be garbage data. Depending on the way this secondary task is implemented into the scene, the resulting gaze data focused on this secondary task might be difficult to differentiate from the main task, possibly skewing the results. It could also interfere with the ability to properly perform the main task itself (i.e., indicating when it is safe to cross the road). Furthermore, similarly to the memorization task, verification of the secondary task needs to be performed.

An auditory secondary task could be used as well. Many different types are possible here, but a commonly used test is a frequency test, where participants have to verbally respond to auditory input, identifying signals being either high or low frequency, compared to an initial sound queue.

Another type of task that can be considered is a counting task to be performed during the videos. When a participant is asked to count forwards or backwards from an arbitrary number, their cognitive load is increased because the brain is processing numerical arithmetic operations (Kazui, Kitagaki, and Mari 2000, Goh, Pearce, and Vas 2021). An operation like this will be unlikely to cause garbage data in the recorded gaze data.

All in all, when considering these secondary tasks, it can be concluded that most of them might interfere with the main task too much. Therefore, the task that implicitly raises the cognitive load, while maintaining focus on the main task, is the one that was opted for, namely the gaze contingency window. The gaze contingency window is also the least interruptive type of secondary task, allowing for a larger number of videos to be shown to the participants in a set amount of time, when compared to these interruptive types of secondary tasks.

2.4. EXPERIMENT

Performing the experiment with the EyeLink® 1000 Plus requires a set-up of two computers. The first is called Host PC, handling all the data collecting of the camera. The second is called Display PC, handling the experiment logic and display of the stimuli. This includes everything from displaying instructions to the participants, to showing inputs for the calibration of the camera. It also includes applying the gaze window on top of the stimuli in real time, using measured gaze data from the Host PC.

The eye-tracker itself consists of an apparatus, holding the high-speed camera (with a 16 mm lens), as well as an infra-red illumination source. Additionally, a head-mount is included in which participants can rest their head while performing the experiment. The head-mount is used to ensure minimal head-movements while measuring the participant's gaze.

While implementing rendered stimuli, it became clear they were not able to be displayed as initially intended. The Display PC was not able to display the 1080p stimuli in real time including the gaze window, resulting in a frame drop rate of over 90 %. The decision was made to re-encode the stimuli to 720p, rather than sacrificing the frame rate and the bit rate. This, however, still resulted in frames being dropped, predominantly at the start of every trial. After padding the trials with the neutral in-between trials screen-color, the dropped frames were mostly mitigated to outside of the actual trial duration. A drawback of this necessary fix is that all stimuli are now 1 s longer, with a new runtime of 11 s.

The specifications of the Display PC are as follows: Intel® Core™ i7-8700 CPU @ 3.20 GHz, NVIDIA® GeForce™ GTX 2080 Ti GPU, using 2 × 8 GB physical memory. The display monitor used is BenQ® ZOWIE™ XL2540. The display resolution was set to match the stimuli at 720p. The highest monitor refresh rate for this resolution was used (144 Hz).

The monitor and eye-tracker need to be placed optimally in order to accurately measure the participants eye movements. First the screen height was set to the eye height of the participants, when resting their head in the chin rest. The camera was placed in between the screen and the participant, without obstructing vision of the monitor. This resulted in the following set distances and dimensions, used by the Host PC to accurately transform the measured gaze to pixel positions on the screen, which are used in the analysis:

• eye-to-monitor-top:	1000 mm
• eye-to-monitor-bottom:	1030 mm
• camera-to-monitor:	543 mm
• screen-width:	544 mm
• screen-height:	303 mm

The experimental set up can be seen in Figure 2.11.

The SR Research Experiment Builder (SR Research Ltd. 2020c) was used to build the logic behind the experiment. All displayed information was designed such that it was readable and showed only necessary information. The background color was chosen such that the average luminance did not change when switching from and to trials. A custom script obtained from the SR Research forums was used to implement a circular

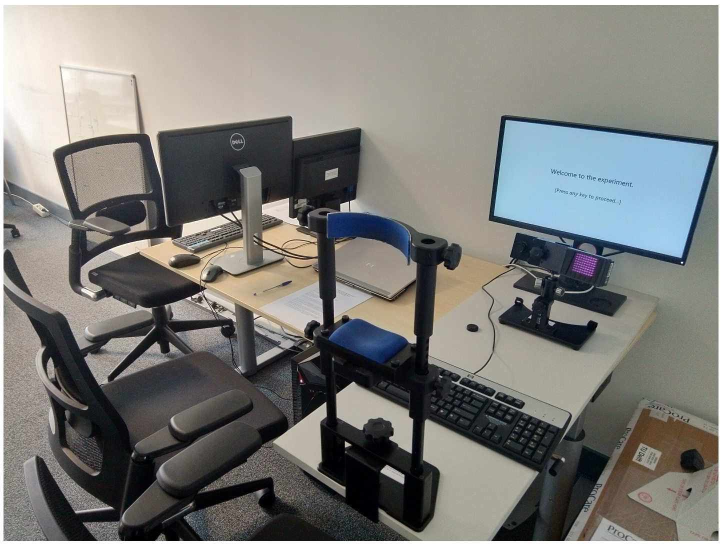


Figure 2.11: Set up used for conducting the experiment. The participant sits in front of the eye-tracker, placing their head in the head-rest.

gaze window (SR Research Ltd. 2020a). This script makes use of an image of a circular mask in combination with the latest gaze measurement in order to project the gaze contingent window at that measurement.

The experiment itself is split into several phases, as recommended by the SR Research Experiment Builder User Manual (SR Research Ltd. 2020d). The flow of the experiment is shown in Figure 2.12.

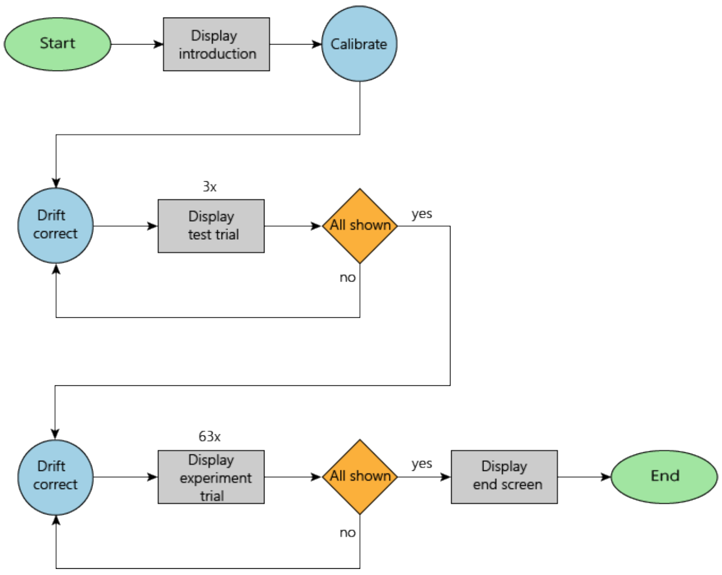


Figure 2.12: Flowchart showing the way the experiment is presented to the participants.

First, an introduction is given to the participant. During this phase, the participant has time to ask questions. After the introduction phase, the participant's eyes are calibrated, using the calibration methods included with the Host PC set up. Afterwards, a demonstration of three trials is given to let the participant familiarize themselves with the traffic situation, the different signal types and the gaze window sizes. It is explained to the participant that they should press the space bar on their keyboard when they deem it safe to cross the road. Between each trial, a drift correction is applied, which allows the camera to recalibrate the position of the eyes to the center of the screen. The participants are allowed to advance through this drift correction themselves, effectively being in control of the experiment flow throughout all the trials.

During the demonstration phase, the participant is still allowed to ask questions. After the demonstration phase is completed, the actual trials start. All trials (of which an overview can be found in Appendix A), are randomized by the Experiment Builder, including a restriction such that two subsequent trials never have the same gaze window size. During this phase, where all 63 trials are presented back-to-back, no interaction between the participant and the supervisor is allowed.

The total experiment duration is estimated between 30 min to 45 min, when taking into account the familiarization phase, as well as small pauses between each trial.

After the experiment is finished, the participants are asked to fill in a questionnaire about the acceptance (self-reported clarity) of the different eHMIs (Van der Laan, Heino, and De Waard 1997). In this scale, nine Likert items (Likert 1932), divided between two dimensions (i.e., usefulness and satisfying) are assessed. This is done for all three signal types.

Finally, the participants are presented with a small reward, which is not communicated to them upfront.

2.5. Analysis

The gaze data recorded with the EyeLink® is stored in the EyeLink Data Format (EDF), which are binary files and store a wide range of recorded and processed signals (SR Research Ltd. 2020b). Since the eye-tracker's processing methods are a black box, a full analysis is performed on the raw gaze data. A script is used to extract only the raw gaze and pupil size metrics (Etter and Biedermann 2018), and store them in a MAT file, a MATLAB® specific file format used to store workspace variables. This makes it easier to load the raw data on subsequent iterations of the analysis.

The response data is stored in a tabular fashion in plain text files (TXT), where each line represents a key-press event.

During the analysis, the data is processed and transformed into useful metrics, namely:

- Reaction time;
- Individual gaze metrics:
  - Saccade count;
  - Saccade amplitude;
  - Fixation duration;
- Dispersion.

For the statistical analysis of the reaction times and individual gaze metrics, trials containing preset 7 (where none of the cars stop, recall Table 2.2), are left out of consideration. These trials are included in the experiment to prevent a learning effect regarding the cars' stopping condition. But when it comes to reaction time measurements, they distort the data. Because no reaction time is expected when no car is stopping, participants either react way later compared to the other presets, or do not react at all. To remain consistent during the entire analysis, the individual gaze metrics for trials from this preset have also been left out of consideration.

2.5.1. Reaction time

The reaction time of the participants for each of the trials follows from the key-press event data.

A couple of assumptions were made in order to avoid getting false positives (e.g., participants accidentally pressing space bar early). For the analysis, only key presses performed after the signal was shown are considered. Additionally, a minimum response time of 0.18 s is assumed (Jain et al. 2015).

2.5.2. Individual gaze metrics

During the processing of the gaze data, a signal analysis script, created by the TUDelft CoR Department is used to filter the signal. It identifies blinks in the signal, by looking for gaps in pupil size data, and interpolates the gaze and pupil size data around these gaps, effectively removing them. It then smooths the signal with a median filter, to remove noise from measurements.

Afterwards, the filtered signals are differentiated and fed to a Savitzky-Golay filter in order to approximate the gaze speed. The filter is meant to adjust for any distortion introduced into the signal by differentiating.

Its result is compared to a certain threshold (2000 $\mathrm{px\,s^{-1}}$) in order to identify saccades and fixations. Furthermore, it is assumed that the minimum saccade duration equals 10 ms, and the maximum saccade duration equals 150 ms. Conversely, a fixation is identified by any period that is not a saccade, with a minimum duration of 40 ms.

2.5.3. DISPERSION

To be able to say something about the degree of focus of the individual participants, instead of actually looking at the individual participants, a measure is created where the whole group is investigated instead. The gaze of the participants as a group can be seen as point cloud that changes over time.

The degree of focus of the group can be expressed in a degree of dispersion of the point cloud. In other words, how close together are all the individual gaze points at any given moment in time?

A metric that could answer this question is the Standard Deviational Ellipse and has been used for quantitative analysis of geographical data before (Lefever 1926, Yuill 1971, Wang, Shi, and Miao 2015). It translates pretty well to the gaze data, since that is also a form of spatial data.

Within a point cloud, there always exists a maximum and minimum variance in a certain direction (Yuill 1971). This direction might not be in the same direction as our coordinate system, but under a certain angle θ. It happens to be that this maximum and minimum are always under 90°, or right-angled.

The angle and the maximum and minimum variance can be calculated. From the variance, the standard deviation in each primary axis can be obtained. To obtain said values, the following approach is used (from Yuill 1971, Wang, Shi, and Miao 2015):

For every point i in the cloud of n data points, a set of coordinates is defined:

$$\mathbf{x}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}. \tag{2.11}$$

The mean of the n data points in the direction of both coordinates are determined:

$$\mathbf{x}_\mu = \begin{bmatrix} x_\mu \\ y_\mu \end{bmatrix}, \tag{2.12}$$

where

$$\begin{cases} x_\mu = \dfrac{1}{n}\displaystyle\sum_{i=1}^{n} x_i \\ y_\mu = \dfrac{1}{n}\displaystyle\sum_{i=1}^{n} y_i \end{cases}. \tag{2.13}$$

Then, the origin of the point cloud is transformed to the mean, which gives new coordinates for all samples, relative to the mean:

$$\bar{\mathbf{x}}_i = \begin{bmatrix} \bar{x}_i \\ \bar{y}_i \end{bmatrix} = \mathbf{x}_i - \mathbf{x}_\mu = \begin{bmatrix} x_i \\ y_i \end{bmatrix} - \begin{bmatrix} x_\mu \\ y_\mu \end{bmatrix}. \tag{2.14}$$

Then, an angle θ is introduced, which rotates the axes around the mean of the point cloud, transforming the coordinates of the data to this new coordinate system. The rotation is performed using a rotation matrix. A standard rotation matrix **R**, rotating a standard Carthesian coordinate system counter-clockwise looks as follows:

$$\mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{2.15}$$

The gaze data is in a non-standard coordinate system, with the origin in the top-left of the screen, going down the screen in positive direction, this means applying the standard rotation matrix **R** means the coordinate system gets rotated clockwise around its origin. This yields the following equations:

$$\mathbf{x\prime}_i = \begin{bmatrix} x\prime_i \\ y\prime_i \end{bmatrix} = \mathbf{R}(\theta)\bar{\mathbf{x}}_i = \begin{bmatrix} \bar{x}_i\cos\theta - \bar{y}_i\sin\theta \\ \bar{x}_i\sin\theta + \bar{y}_i\cos\theta \end{bmatrix}. \tag{2.16}$$

With these transformations applied, variances $\sigma_{x\prime}$ and $\sigma_{y\prime}$ can be determined for any angle θ as follows:

$$\begin{cases} \sigma_{x'}^2 = \frac{1}{n}\sum_{i=1}^{n}(x'_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(\overline{x}_i\cos\theta - \overline{y}_i\sin\theta)^2 \\ \sigma_{y'}^2 = \frac{1}{n}\sum_{i=1}^{n}(y'_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(\overline{x}_i\sin\theta + \overline{y}_i\cos\theta)^2 \end{cases}. \tag{2.17}$$

In order to obtain angle θ where $\sigma_{x'}^2$ is maximum, and $\sigma_{y'}^2$ is minimum (or vice versa), any of the two equations can be differentiated with respect to θ and equated to 0 as follows:

$$\frac{\mathrm{d}\sigma_{x'}^2}{\mathrm{d}\theta} = \frac{2}{n}\sum_{i=1}^{n}(\overline{y}_i^2\sin\theta\cos\theta + \overline{x}_i\overline{y}_i(\cos^2\theta - \sin^2\theta) - \overline{x}_i^2\sin\theta\cos\theta) = 0. \tag{2.18}$$

The general solution to this equation can be obtained by applying the standard formula for finding the roots of a quadratic equation:

$$ax^2 + bx + c = 0 \implies x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}. \tag{2.19}$$

Applying the general solution to Equation 2.18 yields the following θ's:

$$\theta_{1,2} = \arctan\left(\frac{(\Sigma_{i=1}^n \overline{x}_i^2 - \Sigma_{i=1}^n \overline{y}_i^2) \pm \sqrt{(\Sigma_{i=1}^n \overline{x}_i^2 - \Sigma_{i=1}^n \overline{y}_i^2)^2 + 4(\Sigma_{i=1}^n \overline{x}_i\overline{y}_i)^2}}{2\Sigma_{i=1}^n \overline{x}_i\overline{y}_i}\right). \tag{2.20}$$

Since the solutions are always orthogonal to each other, obtaining only one of the solutions is sufficient. This simplifies the solution as follows:

$$\theta = \arctan\left(\frac{A+B}{C}\right), \tag{2.21}$$

with

$$\begin{cases} A = \sum_{i=1}^{n}\overline{x}_i^2 - \sum_{i=1}^{n}\overline{y}_i^2 \\ B = \sqrt{(\sum_{i=1}^{n}\overline{x}_i^2 - \sum_{i=1}^{n}\overline{y}_i^2)^2 + 4(\sum_{i=1}^{n}\overline{x}_i\overline{y}_i)^2} \\ C = 2\sum_{i=1}^{n}\overline{x}_i\overline{y}_i \end{cases}. \tag{2.22}$$

Now $\sigma_{x'}^2$ and $\sigma_{y'}^2$ can be determined by inserting θ (obtained from Equation 2.21) into Equation 2.17.

$\mathbf{x}_\mu$, θ, $\sigma_{x'}^2$ and $\sigma_{y'}^2$ give great insight into how the gaze of the participants as a group is spread, as the mean gaze $\mathbf{x}_\mu$ tells something about the average point of interest. θ, $\sigma_{x'}^2$ and $\sigma_{y'}^2$ show the degree of dispersion, (or inversely, the degree of focus), in two dimensional space. It shows along which axis the variance is the maximum, and along which axis it is minimum.

ECCENTRICITY

A second metric that was investigated follows directly from the dispersion. The eccentricity of an ellipse is a characteristic that describes the roundness of the ellipse. The lower the eccentricity is, the more the ellipse resembles a true circle. The idea here is to calculate the eccentricity over time, to see if and when the ellipse nears the shape of a circle, possibly indicating a single point of focus (or at the very least indicating a similar variance in the principle axes of the ellipse) of the group of participants.

The eccentricity e of the ellipse is described by the following equation:

$$e = \sqrt{1 - \frac{\sigma_{y'}^2}{\sigma_{x'}^2}}. \tag{2.23}$$

ELLIPSE AREA

Going from the dispersion, the maximum and minimum variance of the point cloud could be reduced even further into a single metric: the area of an ellipse. An ellipse can be described by two axes and an angle. By taking the maximum standard deviation as the major axis, and the minimum standard deviation as the minor axis, there is enough information to describe its area:

$$\begin{aligned} A_e &= \pi\sqrt{\sigma_{x'}^2\sigma_{y'}^2} \\ &= \pi\sigma_{x'}\sigma_{y'} \end{aligned}. \tag{2.24}$$

In order to scale to a prescribed confidence interval, a magnification factor needs to be applied to both standard deviations (Wang, Shi, and Miao 2015). For two dimensional data, a scale factor $f_{2d,95} = 2.4477$ has been approximated, in order to obtain a confidence interval of 95 % (Wang, Shi, and Miao 2015). This turns Equation 2.24 into:

$$A_{2d,95} = f_{2d,95}^2 A_e = f_{2d,95}^2 \pi\sigma_{x'}\sigma_{y'}. \tag{2.25}$$

Take note that, in order to draw this ellipse, information such as its origin $\mathbf{x}_\mu$ and angle θ of the ellipse with respect to the screen's coordinate system are also required.

2.5.4. PROCEDURE

MATLAB® (The MathWorks, Inc. 2022) is used perform this analysis. Pseudo-code shown in Listing 1 shows the full structure of used code. All scripts and functions used to perform the analysis and produce all figures shown in Chapter 3 can be found in Appendix C.1.

```
run Main_Init.m
    def CONSTANTS

run Main_Import.m
    call ImportFromEdfs()
        call Edf2Mat()
    call ImportFromTxts()

run Main_PrepareData.m
    call PrepareEdfData()
        call ProcessEyes()
    call PrepareTxtData()
        call AccumulateKeyPresses()
    call GetResponseTime()
    call PrepareHeatMap()
    call CalculateDispersion()
    call CalculateEccentricity()
    call CalculateEllipseArea()
    call PrepareBoxChartData()

run Main_Statistics.m
    call StatisticalTests()

run Main_Plot.m
    call PlotRawVsPrepared()
    call PlotHeatMaps()
    call PlotResponseTimes()
    call PlotEllipseSize()
    call PlotEccentricity()
    call PlotBoxCharts()
    call PlotScatterPlots()

run Main_Redraw.m
    call InterpolateData()
    call SampleKeyPresses()
    call DrawGazeOnStimulus()
    call ReconstructTrial()
```

Listing 1: Pseudo-code showing the structure of the MATLAB analysis.

INITIALIZE SCRIPT

During this part of the procedure, global constants are defined, which are used throughout the rest of the scripts' functions. This includes but is not limited to information about the number of participants, the number of trials and the independent variables. This is also the place where several important directories are set (e.g., input and output directories).

IMPORT FILES

In this section of the script, the gaze and reaction time data is imported. First, all of the participants' recorded gaze and pupil size data is imported. After obtaining the correct signals for each participant, they are resorted to match the trial numbers, and stored into a structure. This structure is then saved to a separate MAT file, to make importing on subsequent runs faster.

From the TXT files, the response data is processed into a table. For each participant, a cell of the table holds an array of all key-presses along with their respective time stamps that participant made throughout the experiment.

PREPARE DATA

The raw signals are prepared into workable arrays. Since there is a slight variation in trial length, introduced by the Display PC's run time precision during the experiment, some of the trials are a few samples short of the expected length, while others are a few too long. The length is equalized by discarding all samples exceeding the expected run-time for each trial, and padding empty samples to trials that do not reach the expected run-time.

The individual gaze metrics are then prepared as explained previously in Subsection 2.5.2.

The entries from the raw table produced by importing the TXT files are split and resorted by trial.

Additionally, the key press data is accumulated over time for each trial. This gives insight into when the keys were pressed for each trial specifically. Whenever a participant presses the key, the signal increases. When a key is released, the signal decreases again. This way, a stair-step function is produced for the duration of each trial.

To gain insight in the differences in key-press data between conditions, a similar accumulation is performed, but instead, the trials belonging to the same condition are grouped. This results in six signals which can easily be compared to see differences between the different conditions.

Afterwards, the reaction time metric is calculated as previously explained in Subsection 2.5.1.

To give some more insight into the data, several heat maps are prepared. One for each of the trials, combining all of the participants' gaze per trial, used to see if there are anomalies in specific trials. Additionally, two heat maps of all trials combined are created: one from the raw gaze data, and one from the prepared gaze data. They are used to see if and how the preparation of the data changed the overall gaze. The resulting matrices were normalized for a better comparison.

To make meaningful box charts and scatter plots (and performing a statistical analysis on them), the dataset containing values for all participants for each individual trial needs to be reduced, to single values for each participant for each condition.

This means that for each metric to be analyzed, arrays are created that average the results of that metric per condition, such that, when plotted later, the box-chart consists of n values, where n is the number of participants. The values inside are the averaged conditions.

In order to analyze the within-group effects of the independent variables, all data points for each independent variable are grouped together separately, resulting in averages for separate condition (i.e., one average value for each participant for each signal type, and one average value for each participant for each gaze window size).

In order to analyze the between-group effects of the independent variables, again averages are created for the trial data of each participant. Only this time, the grouping of the variables is done differently. Instead of one average value for each participant for each condition, an average value for each *combination* of independent variable conditions is created.

STATISTICS

The resulting data-sets, used for analysis on the within- and between-group effects, are summarized and exported into tables. For a given data-set, the summary includes the mean, sum, min, max, range, median, variance, standard deviation and amount of data points for each condition.

Then, statistical tests are performed on both of the independent variables individually (looking at within-group effects), as well as the combination of the independent variables (looking at between-group effects.

The test that is being performed is an ANOVA (analysis of variances), after which a post-hoc multi-compare is performed, in this case a pairwise t-test with a Bonferroni correction. The correction is necessary to protect against Type-1 errors (i.e., falsely rejecting the null hypothesis for the test when it should not have been) (S. Lee and D. K. Lee 2018).

The type of ANOVA that is performed depends on what is investigated. For the analysis of the within-group effects, a one-way ANOVA was performed. For the analysis of the between-group effects, an n-way ANOVA was performed.

PLOT

This section of the code includes all the scripts used to create figures out of the calculated metrics. First, a comparison between the raw and processed eye-tracker data is created. Then, heat maps of the combined gaze for all trials are created, again, showing a comparison between the raw and processed data. Afterwards, the accumulated response times grouped per condition are created. A similar figure showing the change of the dispersion over time is created, by plotting the average size of the ellipse grouped per condition over time. The same is done for the eccentricity of the ellipse. Finally, box charts and scatter plots for the reaction time and each of the individual gaze metrics are created, giving a visual representation of the results of the statistical analysis.

REDRAW

This part of the code is optional, and was created to give an even greater insight into the results.

First, the individual gaze data is transformed into 60 fps, to match the trials' frame rate.

A sampled version of the reaction time metric is created to get the key-press status per participant at each frame of the trial duration.

Then, the gaze data is used to draw markers on each of the trials, creating a swarm of gaze markers throughout the trials. The sampled reaction time can be used as an indicator for each marker to show whether or not a participant has performed a key-press.

Additionally, the dispersion ellipse can be drawn on the trials as well, giving even more information.

A second functionality presented in this part is the ability to redraw each specific individual trial, including gaze-contingent window. In practice this is only used to recreate an example of the experience of the participants.

2.5.5. QUESTIONNAIRE

The results of the Van der Laan-questionnaire each participant filled in, are manually copied from the print-outs into Excel. The Likert items are transformed into values ranging from -2 to 2.

Then, MATLAB® is used to import the resulting file, and create box charts of the included data. A comparison between the different signal types is shown for the *usefulness* and *satisfying* dimensions of the test, as well as for each of their individual components.

Additionally, a radar plot is created as well, for a more comprehensive comparison between the individual components (Yoo 2022).

3 Results

This chapter aims to explain the experiment's results in-depth. First, the general response is investigated. This means the raw gaze data over time is investigated. A comparison between the raw and the prepared signal is performed. Second, the key-press data is investigated. This includes a statistical analysis. Third, an analysis similar to the key-press data is performed on the saccade and fixation metrics. Fourth, the dispersion is analyzed with a novel approach, first, over time, and second, quantitatively. Finally, the results of the self-reported questionnaire are presented.

For a visual reference, figures are included for every metric. All values mentioned throughout this chapter are listed in a complete summary in Appendix B.

The experiments were conducted over a span of 2 weeks. There were 23 participants (18 male, 5 female), ranging from 21 to 67 years old. The average age of the participants was 32 years old.

3.1. General response

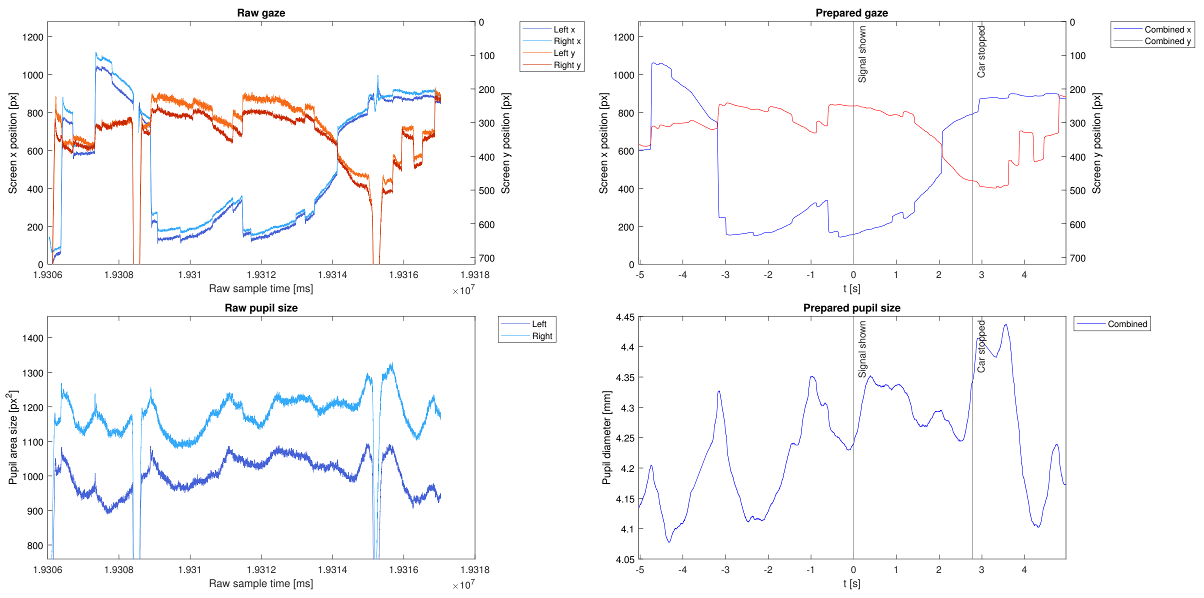
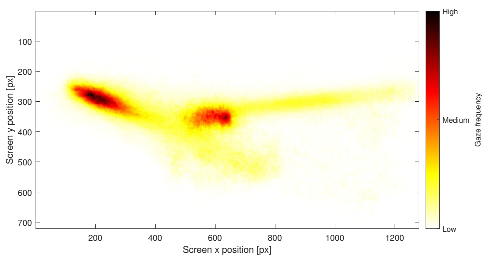
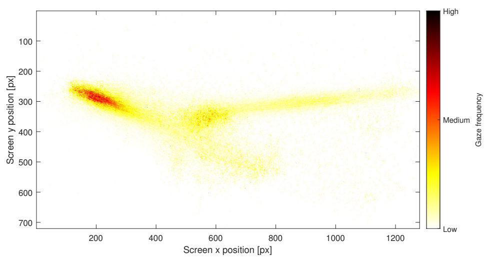


Figure 3.1: Example of raw versus prepared data. The data shown is from a single trial of one participant.

Figure 3.1 shows an example of the general response of a single trial of one participant. Each eye has a separate recorded signal for both the screen's x and y position (measured in pixels (px) from the top left position), and a raw pupil size, measured in square pixels (px^2). The prepared signals, used for further analysis, are also shown.



(a) Heat map of the raw gaze data.



(b) Heat map of the prepared gaze data (filtered).

Figure 3.2: Normalized heat maps of the raw gaze data and the prepared gaze data.

The preparation process removed blinks from the raw signal and smoothed the result. Notice that the two signals for each eye have been reduced to a single, generalized, average signal. This goes for both the gaze and pupil size data. All further analysis is performed on these prepared signals.

A comparative heat map is also produced from all the gaze data. This can be seen in Figure 3.2. Figure 3.2a shows the heat map of the raw gaze data. This includes the extra padded second that was at the start of each trial. Figure 3.2b shows the heat map of the prepared gaze data, where the padded second has been removed. The smoothing done by the processing script created some artifacts in the gaze data. Those were filtered out in this image, but are kept in the rest of the analysis. What can be seen is that the high intensity blob in the center of the screen has disappeared in the prepared gaze data. Furthermore, the image is generally sharper than the raw image, since the noise has been smoothed out of the data.

From these examples a quantifiable analysis cannot be made. However, it is still useful to make comparisons like these. It serves as a good sanity check to see if the data is correct. Among other things, it allows us to see if stimuli are matched correctly and if blinks are removed correctly.

3.2. REACTION TIME

As previously explained, all participants were asked to press and hold the space bar when they deemed it safe to cross the road half-way. The expected result here is that participants did not find it safe to cross when cars are still approaching, and press the space bar when it was clear that the car was stopping.

The cumulative result of this task for each grouped condition can be seen in Figure 3.3.

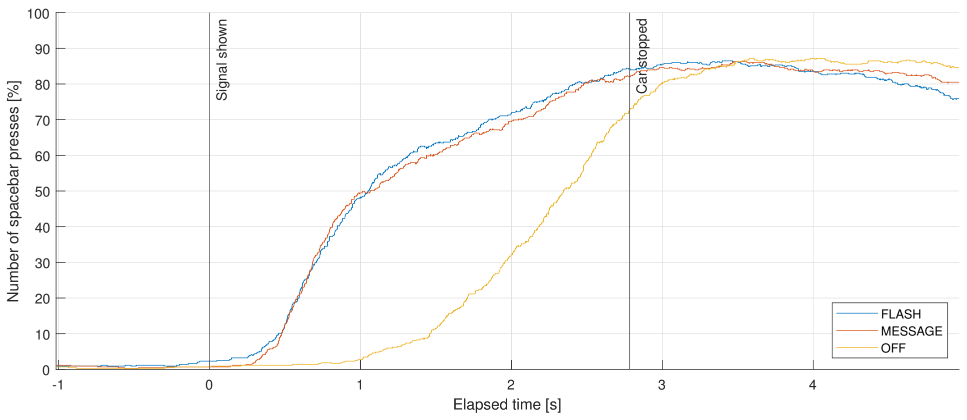
From Figure 3.3a it is apparent that for the signal type the off condition is very different compared to both the flash and message conditions. At the moment the signal is shown, no response can be observed for any of the conditions. Then, within half a second after the signal was shown, a steep incline can be observed for both the flash and message conditions, after which it flattens until the car has come to a full stop. For the off condition, this steep incline is much more gradual and only starts after ~ 1 s after the car started to decrease speed.

When the car has come to a full stop, there is still a $\sim 10\,\%$ difference in the number of key-presses between the off condition and the flash and message conditions.

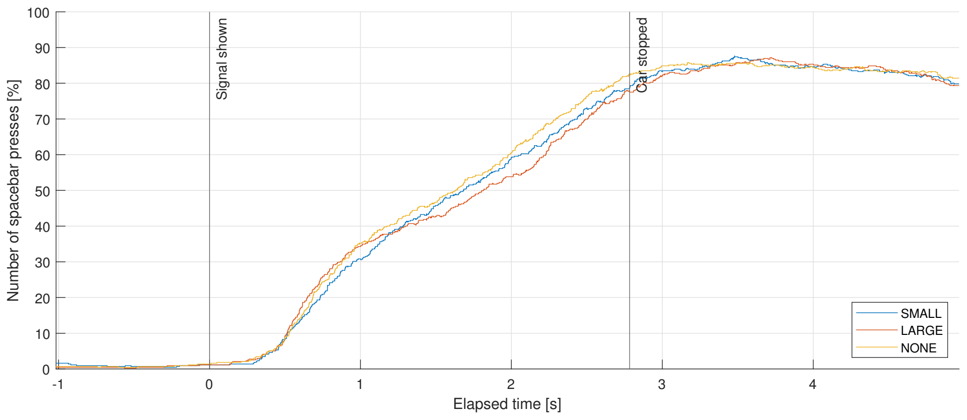
Between the flash and message conditions there is no clear difference. To get a better insight into the response, an analysis is performed on the individual average response times per condition. The key-presses over time for each trial are reduced to a single response time. The first key-press after the signal, taking into account a minimum reaction time, is interpreted as that trial's reaction time. This results in the distributions shown in Figures 3.4 and 3.5, where each observation represents the average response time for a participant for that groups' condition (including all conditions of the other group). The average of the participants' averages is also displayed.

Looking at the signal type, these results show a clear distinction in reaction time for the off condition, in comparison to the flash and message conditions. The median response times for the flash and message conditions are ~ 1.11 s ($\overline{x} \approx 1.33$ s, $\sigma \approx 0.77$ s), and ~ 0.96 s ($\overline{x} \approx 1.34$ s, $\sigma \approx 0.81$ s), respectively. For the off condition, the median response time is ~ 2.23 s ($\overline{x} \approx 2.31$ s, $\sigma \approx 0.60$ s). This is over twice as slow as the other conditions. From Figure 3.5a it can even be seen that every participant has a faster average response time for the message and flash conditions, compared to the off condition.

This slow response time for the off condition is also reflected in the minimum and maximum response



(a) Cumulative number of space bar presses over time for all trials grouped by the signal type conditions.



(b) Cumulative number of space bar presses over time for all trials grouped by the gaze window size conditions.

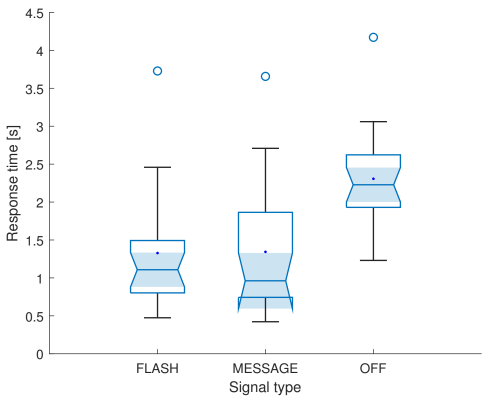
Figure 3.3: Cumulative number of space bar presses over time for all trials. The values are presented as a percentage of the maximum possible number of space bar presses at any given time.

times. For the off condition the minimum response time is ~ 1.23 s, and the maximum response time is ~ 4.17 s. For the flash condition the minimum is ~ 0.47 s and the maximum is ~ 3.73 s. For the message condition the minimum is ~ 0.42 s and the maximum is ~ 3.66 s.

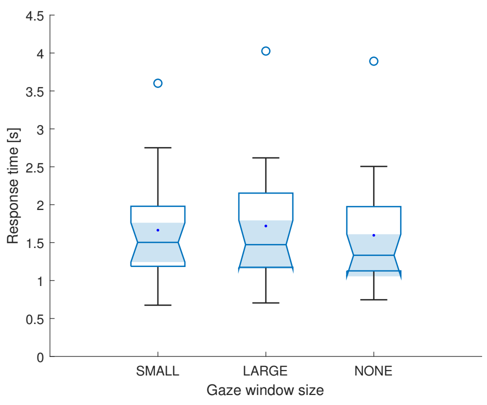
Following the results from the 1-way ANOVA, a post-hoc test using Bonferroni's critical value type was performed, to find that the difference in response time between the flash and off conditions is significant ($p < 0.001$) at the 95 % confidence interval. The difference between the message and off conditions is also significant ($p < 0.001$) at the 95 % confidence interval. The difference in response time between the message and flash conditions is not significant ($p \approx 0.977$) at the 95 % confidence interval.

For the gaze window size conditions, the response times look very similar as can be seen in Figures 3.4b and 3.5b, where the median response times range from ~1.33 s to ~1.50 s ($\bar{x} \approx$ 1.60 s to ~1.72 s, $\sigma \approx$ 0.66 s to ~0.76 s). Following the results from the 1-way ANOVA, followed by Bonferroni's critical value type post-hoc test, the difference in response time of none of the pairs' means are significantly different.

To inspect the effect of one condition on the other (i.e., the effect of gaze of the gaze window size on the signal type and vice versa), an n-way ANOVA might give some insights. The results of the condition combinations were averaged, and an n-way ANOVA was performed. The distribution of the results can be seen in Figure 3.6. The results of the n-way ANOVA were analyzed with a post-hoc test with a Bonferroni correction. The results of this test do not show any significant differences in the means between of the signal types with different gaze windows. This is also what the figure suggests. The means are relatively close together, and there are no clear trends visible.

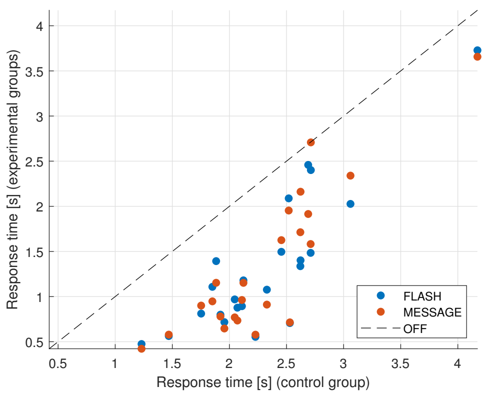


(a) Average response time of each participant for all trials grouped by the signal type condition.

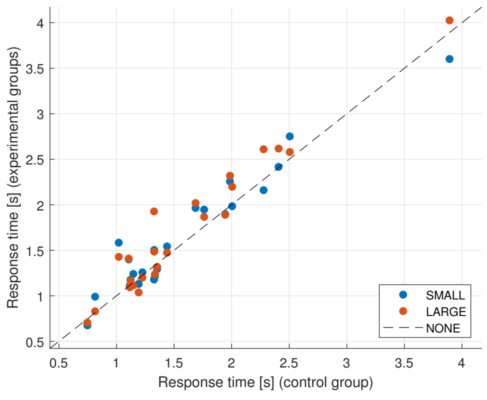


(b) Average response time of each participant for all trials grouped by the gaze window size condition.

Figure 3.4: Average response time per group for each condition. The response times are defined as the delay in response from the time the signal is shown.



(a) Average response times of the experimental groups against the control group of the signal type condition.



(b) Average response times of the experimental groups against the control group of the gaze window size condition.

Figure 3.5: Average response times of experimental groups against control group for each condition. The response times are defined as the delay in response from the time the signal is shown. Results under the dashed line can be interpreted as results faster than the control group. Conversely, results above the dashed line are slower than the control group.

3.3. Saccades and fixations

Throughout the length of a trial, a participant's gaze can be split up in saccades and fixations. A saccade is a section of the gaze data where the participant is moving their attention from one point to another. Its amplitude is defined as the distance between the start of the saccade (the old fixation point) and the end (the new fixation point). A fixation is defined as the time between two saccades. During this period, the participant is fixated on a single point of interest.

3.3.1. Saccade count

Since saccades and fixations alternate, it is unnecessary to count both. As a metric, the saccade count is chosen, since the trials are preceded by a fixation task: a drift correction. This means the number of fixations is always the same or one plus the saccade count (true when the trial ends in a fixation state). Figures 3.7 and 3.8 show the spread of the saccade count for each condition group.

It can be seen that the signal type conditions do not show any significant differences between them. This is supported by the performed paired t-tests, that do not show a significant difference at the 95 % confidence interval.

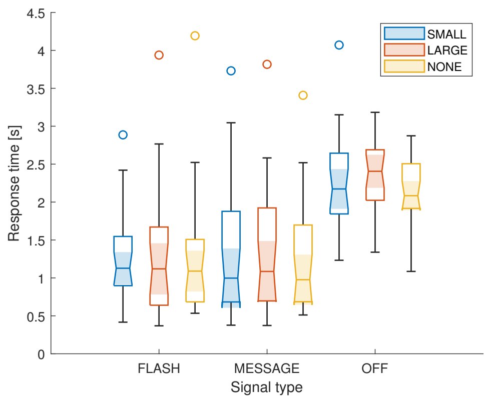
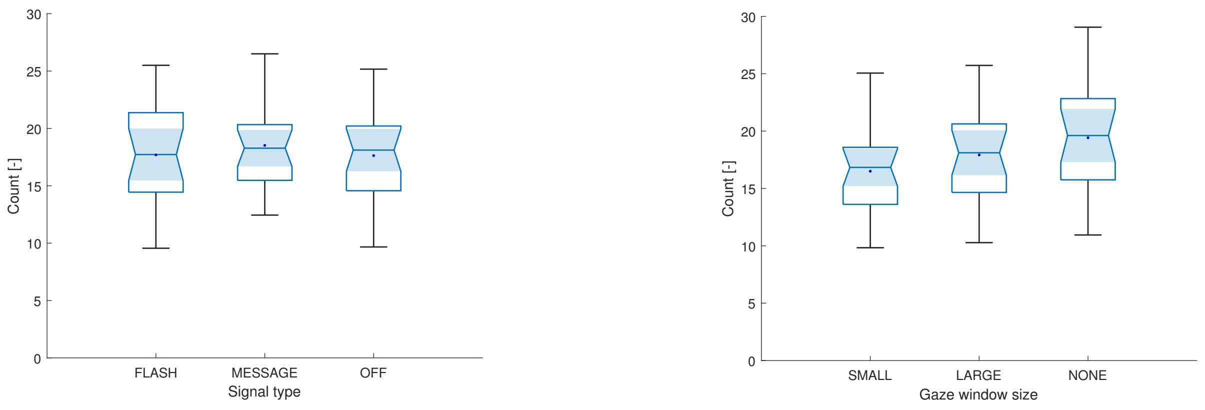


Figure 3.6: Average response times of each participant for all trials, showing the between-group effects of the independent variables (signal type and gaze window size).

For the gaze window conditions, a trend is visible where the saccade count reduces, when the gaze window size gets smaller. The small gaze window size has the lowest saccade count with a median of ~ 16.83 ($\overline{x} \approx 16.50$, $\sigma \approx 3.62$), followed by the large gaze window size, with a median of ~ 18.11 ($\overline{x} \approx 17.92$, $\sigma \approx 3.87$), and finally no gaze window, with a median of ~ 19.61 ($\overline{x} \approx 19.42$, $\sigma \approx 4.62$).

While this seems to be a clear trend, none of the means of the conditions are significantly different at the 95 % confidence interval ($p = 0.053$ being the lowest between the small and none condition).



(a) Average saccade count for all trials grouped by the signal type condition.

(b) Mean saccade count for all trials grouped by the gaze window size condition.

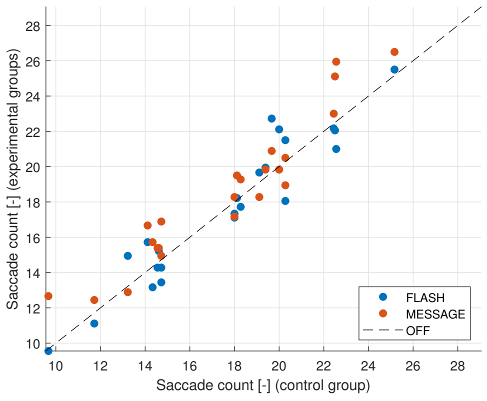
Figure 3.7: Average saccade count per group within each condition, measured over the entire stimulus duration.

Again, the effect of the gaze window size on the signal type is investigated. The average saccade counts of all the participants of the 9 different condition combinations are compared with each other. The result can be observed in Figure 3.9. The same trend visible in the within-group analysis of the gaze window size is visible in this between-group analysis. It shows a downward trend in saccade count when decreasing the gaze window size for all three signal types. An n-way ANOVA was performed on the results, and a post-hoc test with Bonferroni correction on the ANOVA results. These show that this visible trend is not significant.

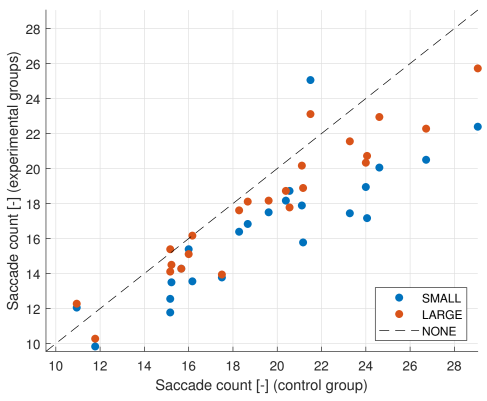
3.3.2. SACCADE AMPLITUDE

The saccade amplitude depicts the distance traveled during the saccade. Here, the saccade amplitude is summed over each trial, resulting in a total distance traveled for each trial. Then it was averaged over the trials for each condition, giving an averaged total distance traveled for each condition for each participant. These results are visible in Figures 3.10 and 3.11.

The average saccade amplitudes for the signal type conditions look fairly similar, but when looking at



(a) Average saccade count of the experimental groups against the control group of the signal type condition.



(b) Average saccade count of the experimental groups against the control group of the gaze window size condition.

Figure 3.8: Average saccade count per group, measured over the entire stimulus duration. Results under the dashed line can be interpreted as results with fewer saccades than the control group. Conversely, results above the dashed line have more saccades than the control group.

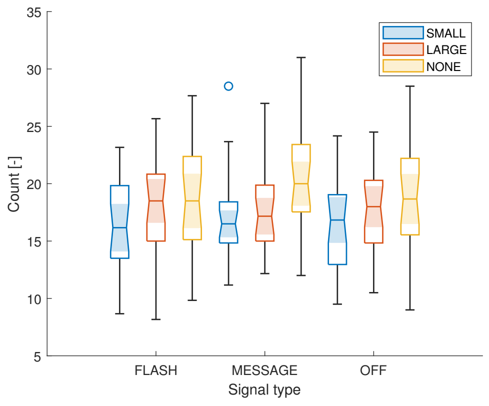
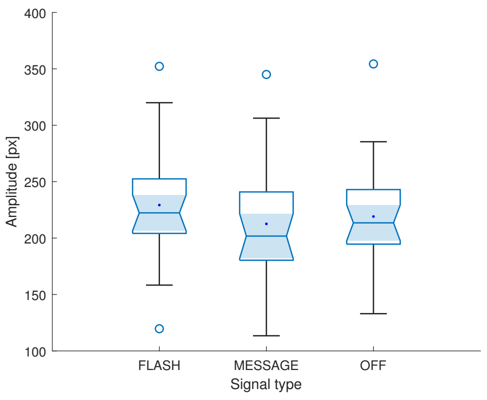


Figure 3.9: Average saccade count of each participant for all trials, showing the between-group effects of the independent variables (signal type and gaze window size).

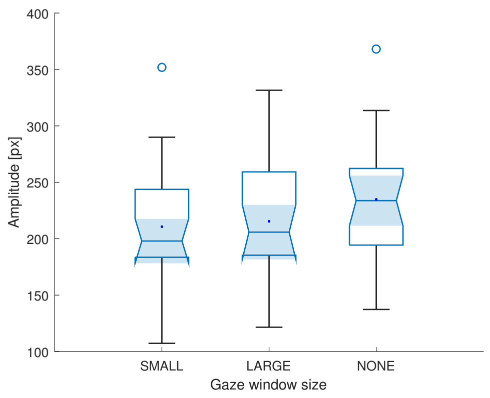
the distribution of the measurements in the scatter plot, it looks like the flash condition is higher, and the message condition lower, when compared to the off condition. The flash condition has a median value of ~ 222.33 px ($\bar{x} \approx 229.26$ px, $\sigma \approx 52.51$ px), compared to the message condition, with a median of ~ 201.78 px ($\bar{x} \approx 212.60$ px, $\sigma \approx 52.04$ px) and off, with a median of ~ 213.48 px ($\bar{x} \approx 219.12$ px, $\sigma \approx 50.98$ px). The one-way ANOVA performed on these results did not hint at any significant differences between the means. The post-hoc test with Bonferroni correction confirmed this.

For the gaze window size, when looking at Figure 3.10b, the large and small conditions look very similar, whereas the control condition looks slightly different. When looking at Figure 3.11b, it can be seen that most participants have a lower average saccade amplitude for the small and large gaze window conditions compared to the none condition. The median for the small condition is ~ 197.94 px ($\bar{x} \approx 210.65$ px, $\sigma \approx 55.22$ px). For the large condition the median is ~ 205.81 px ($\bar{x} \approx 215.43$ px, $\sigma \approx 49.16$ px). For the none condition the median is quite a bit higher, at ~ 233.76 px ($\bar{x} \approx 234.90$ px, $\sigma \approx 53.85$ px). However, the one-way ANOVA performed on these results did not show any significant differences between the means. The post-hoc test with Bonferroni correction confirmed this.

To see if there is any effect of the gaze window size on the signal type, the average of the saccade amplitudes of all the participants of the nine different condition combinations are compared. The result can be observed in Figure 3.12. The results look fairly equal across the board, with only the none gaze window raising the amplitude of the flash and off conditions slightly. An n-way ANOVA was performed on the results.

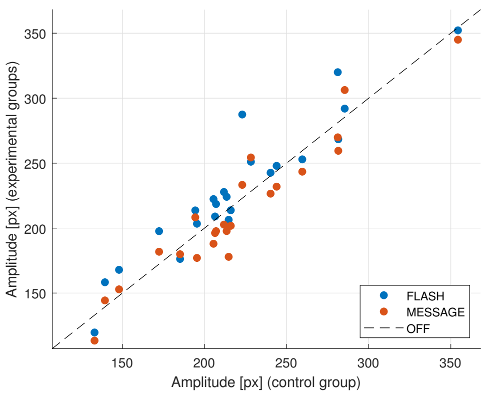


(a) Average saccade amplitude for all trials grouped by the signal type condition.

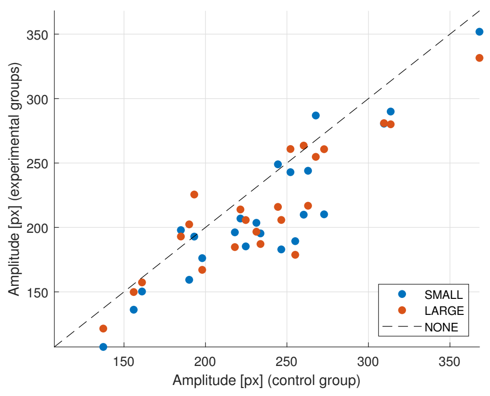


(b) Mean saccade amplitude for all trials grouped by the gaze window size condition.

Figure 3.10: Average saccade amplitude per group within each condition, measured over the entire stimulus duration.



(a) Average saccade amplitude of the experimental groups against the control group of the signal type condition.



(b) Average saccade amplitude of the experimental groups against the control group of the gaze window size condition.

Figure 3.11: Average saccade amplitude per group, measured over the entire stimulus duration. Results under the dashed line can be interpreted as results with smaller saccade amplitudes than the control group. Conversely, results above the dashed line have larger saccade amplitudes than the control group.

The ANOVA and post-hoc test with Bonferroni correction do not show any evidence of the gaze window size having a significant effect on the means of the saccade amplitude between the different signal types.

3.3.3. FIXATION DURATION

The fixation duration is the amount of time a participant fixated on a certain point on the screen. The durations of all trials were averaged for each condition per participant. The results can be seen in Figures 3.13 and 3.14.

It can be seen from Figure 3.13a that the spread on the message condition was considerably smaller than the flash and off conditions, which in turn look very similar. The flash condition has a median of ~ 578.77 ms ($\bar{x} \approx 605.73$ ms, $\sigma \approx 147.30$ ms), the message condition has a median of ~ 556.78 ms ($\bar{x} \approx 573.31$ ms, $\sigma \approx 120.01$ ms), and the off condition has a median of ~ 561.36 ms ($\bar{x} \approx 598.75$ ms, $\sigma \approx 128.08$ ms). A one-way ANOVA was performed on the results, followed by a post-hoc test with Bonferroni correction. These did not show any significant difference between the condition means.

In Figure 3.13b and 3.14b the results of the gaze window size can be seen. A trend can be observed indicating a higher fixation duration when the gaze window gets smaller. The none condition has a median of ~ 508.85 ms ($\bar{x} \approx 557.33$ ms, $\sigma \approx 146.27$ ms), the large condition has a median of ~ 568.57 ms ($\bar{x} \approx 583.66$ ms, $\sigma \approx 123.07$ ms), and the small condition has a median of ~ 599.10 ms ($\bar{x} \approx 636.80$ ms, $\sigma \approx 131.87$ ms). The one-

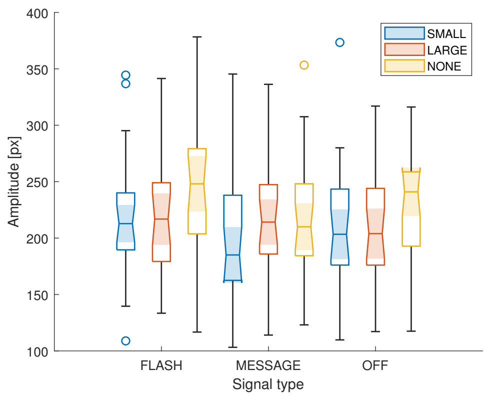
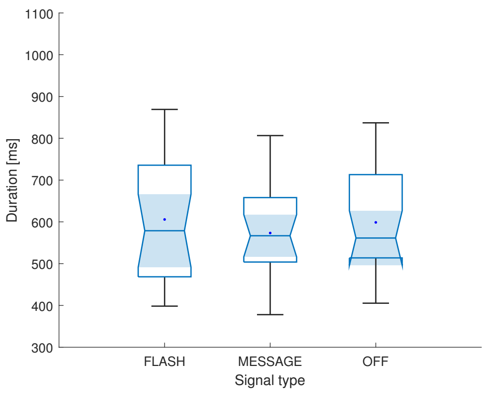
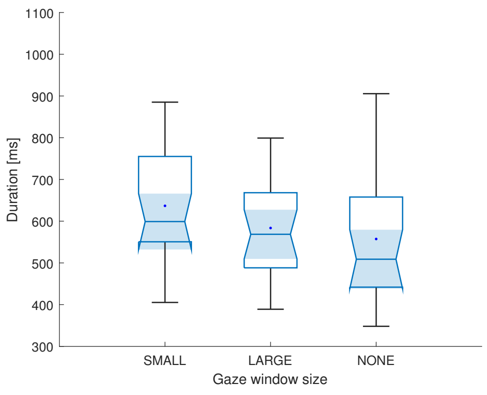


Figure 3.12: Average saccade amplitude of each participant for all trials, showing the between-group effects of the independent variables (signal type and gaze window size).

way ANOVA performed on its results however do not indicate a significant difference in the condition means. A post-hoc test with Bonferroni correction confirms these findings.



(a) Average fixation duration for all trials grouped by the signal type condition.



(b) Average fixation duration for all trials grouped by the gaze window size condition.

Figure 3.13: Average fixation duration per group within each condition, measured over the entire stimulus duration.

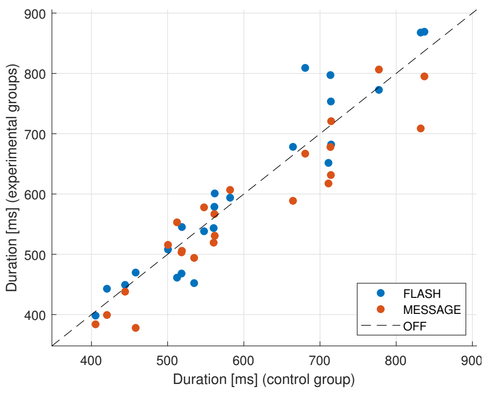
To see if there is an effect of the gaze window size on the signal type, the combined conditions are compared. The result can be seen in Figure 3.15. The same trend visible in the gaze window size can be observed here for each signal type separately, indicating no effect between the groups. An n-way ANOVA was performed, with a post-hoc test with Bonferroni correction. This test indicated no significant differences.

3.4. DISPERSION

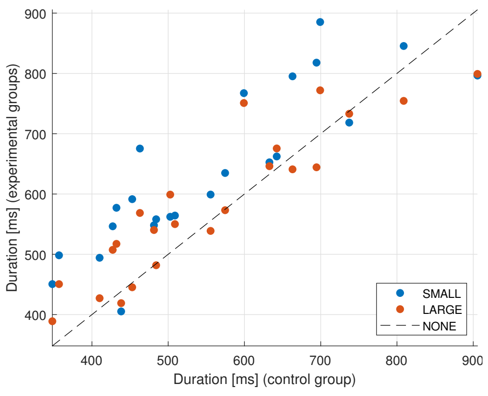
The dispersion, which can be seen in Figure 3.16, is a quantitative metric. It shows the area of the ellipse, encapsulating all the gaze positions at a time at the 95 % confidence interval, and is drawn as a percentage of the maximum possible ellipse. A low percentage indicates that the gazes of the participants are clustered together. On the contrary, a high percentage indicates that the gazes of the participants are dispersed.

A small dip can be observed right at the beginning. This is where the video is loaded, and it appears to draw the focus of all the participants to a similar point, resulting in a low dispersion. Then, throughout the trial, the gaze cloud gets dispersed quite significantly, above 12 %. Due to the nature of how the trials were set up, the gazes get slightly focused again, after they disperse again.

Then, when the signal is shown, the dispersion drops to its lowest point, similar to the start of the trial, indicating the attention of the participants was drawn to the same area. After this initial draw, the focus disperses again, but differently between the different conditions.



(a) Average fixation duration of the experimental groups against the control group of the signal type condition.



(b) Average fixation duration of the experimental groups against the control group of the gaze window size condition.

Figure 3.14: Average fixation duration per group within each condition, measured over the entire stimulus duration. Results under the dashed line can be interpreted as results with shorter fixation durations than the control group. Conversely, results above the dashed line have longer fixation durations than the control group.

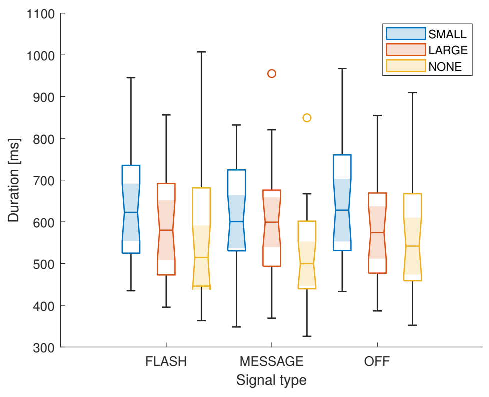


Figure 3.15: Average fixation duration of each participant for all trials, showing the between-group effects of the independent variables (signal type and gaze window size).

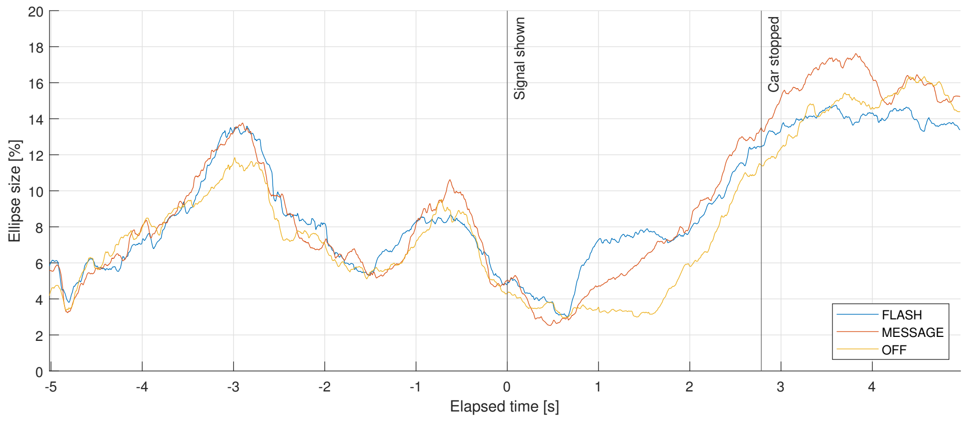
For example, Figure 3.16a shows the dispersion for the signal type conditions. It can be seen that the focus remains focused a lot longer for the off condition compared to the message and flash conditions. For the flash condition, it disperses within 1 s after the signal was shown, while for the message condition, it takes almost 2 s to get up to the same level of dispersion.

A similar, but smaller effect can be seen in Figure 3.16b, for the gaze window size condition. Initially the gaze focuses, but then for the condition without a window, the gaze disperses the fastest. After ~ 2 s the dispersion is nearly identical for the three conditions.

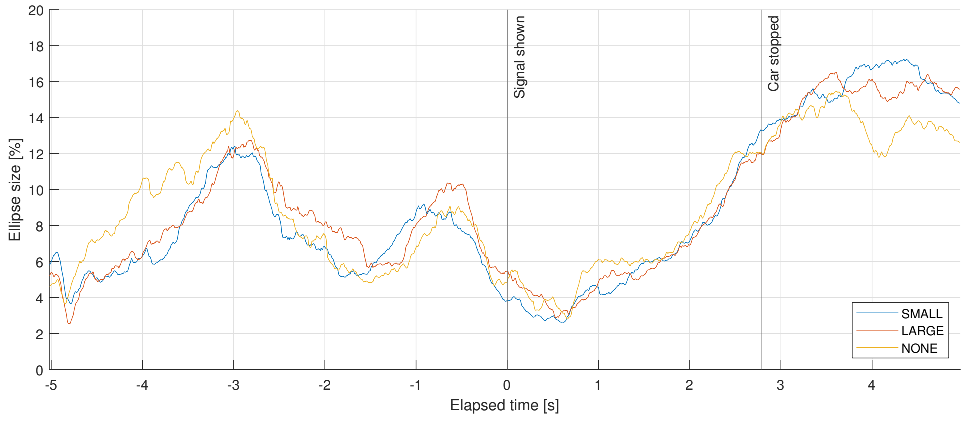
3.5. SELF-REPORTED CLARITY

The results of the Van der Laan-questionnaire (Van der Laan, Heino, and De Waard 1997) and its individual components can be seen in Figures 3.17 and 3.18.

Between the flash and message conditions, the largest difference can be seen in the satisfying scale, as apparent from Figure 3.17a. From Figure 3.17c it follows that participants are much more positive towards the message type, especially on the *pleasant* and *likable* components. The *likable* component of the flash condition is similar to the off condition. On average, all components of both scales are positive for both the flash and message conditions. As can be seen in Figure 3.18a, the flash and message conditions were regarded similar on the *usefulness* scale, while Figure 3.18b shows that the message condition is regarded as more satisfying across all components, compared to both the flash and off conditions. For the message condition, most components were, on average, largely regarded as positive. For the flash condition, the *likable* component



(a) Average dispersion of the gaze cloud for all trials grouped by the signal type condition.

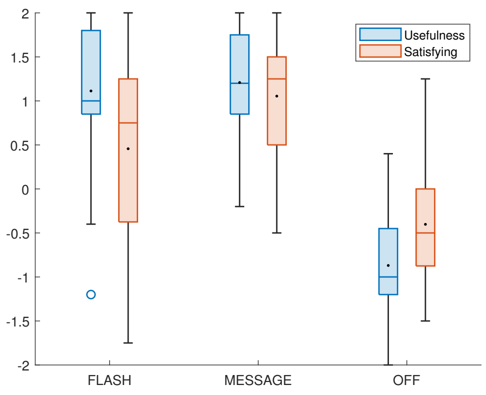


(b) Average dispersion of the gaze cloud for all trials grouped by the gaze window size condition.

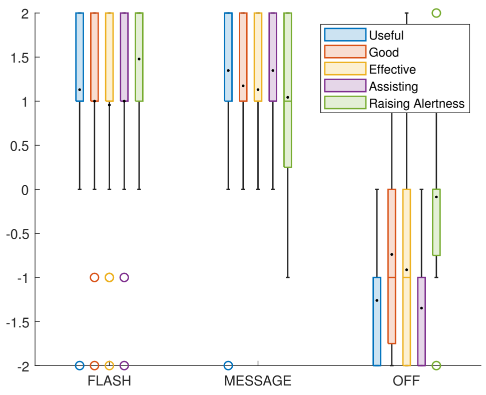
Figure 3.16: Dispersion of the gaze cloud per condition, measured as the area of an ellipse described at the 95 % confidence interval.

was very divisive, with extremes on both the positive and negative side. The rest of the components for the flash condition were mostly positive. This in contrast to the off condition, which was on average regarded as the least satisfying of the three conditions, with most components being mostly negative, except the *likable* component.

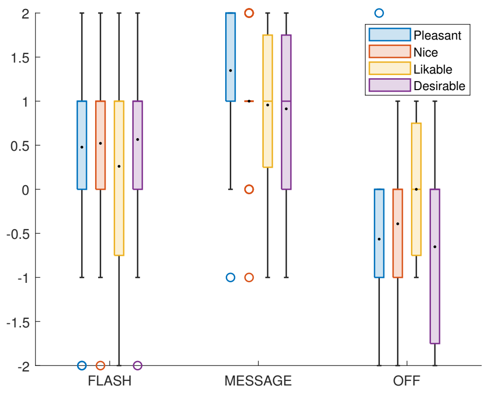
For the off condition it is clear that on both the *usefulness* as *satisfying* scale, the acceptance is really low: lower than both the flash and message conditions. Only the *raising alertness* component of the *usefulness* scale and the *likable* component of the *satisfying* scale are neutral, while all the other components of both scales are negative, as can be seen in Figures 3.18a and 3.18b.



(a) Self-reported Van der Laan acceptance on a Likert scale, showing the usefulness and satisfying results.

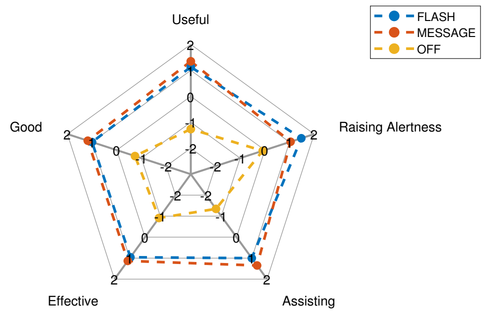


(b) Self-reported Van der Laan acceptance on a Likert scale, showing the individual components of the usefulness component.

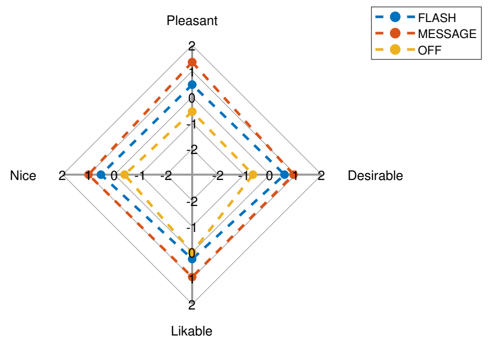


(c) Self-reported Van der Laan acceptance on a Likert scale, showing the individual components of the satisfying component.

Figure 3.17: Self-reported Van der Laan acceptance on a Likert scale, showing both the usefulness and satisfying components.



(a) Means of the usefulness component of the Van der Laan acceptance questionnaire.



(b) Means of the satisfying component of the Van der Laan acceptance questionnaire.

Figure 3.18: Self-reported Van der Laan acceptance on a Likert scale, showing both the usefulness and satisfying components.

4 DISCUSSION AND CONCLUSION

Given the problem of the looming decrease in humanized interaction in traffic, with the increasing prevalence of self-driving cars, research has been conducted to retain some of this interaction. This research project aimed to investigate the effect of the increase in cognitive load on the effectiveness of different signal types displayed on eHMIs of self-driving cars when stopping for a crossing pedestrian. The different signal types were a text-based and a light-based one, both of which have been established as feasible solutions to the problem at hand, but only in fairly limited testing set-ups. Adding various different gaze contingent windows to the equation aimed to investigate if the effectiveness of the different signal types holds in the case of an increased cognitive load.

The main research question this research aims to answer is the following: *What influence does raising the cognitive load for crossing pedestrians have on the effectiveness of text-based eHMIs as opposed to light-based eHMIs and no eHMI?*

To answer this question, an eye-tracking experiment was conducted. 23 participants were shown a range of 63 different trials of 10 s each. This set of trials includes 7 different presets aiming to test the effect of 2 independent variables, each having 2 conditions and 1 control condition. During each trial, the participants were presented a traffic situation consisting of multiple identical self-driving cars, each equipped with an eHMI. In this traffic situation, the participants took the role of a pedestrian wanting to cross the road. When one of the cars yielded for the pedestrian, this car could display a signal on their eHMI, indicating that they were stopping. The participants' task during the experiment was to indicate when they deemed it safe to cross the road, by pressing the space bar. The reaction time of the participants was measured, as well as their gaze during the entirety of the trials at 2000 Hz. This gaze data was transformed into three different metrics: saccade count, saccade amplitude and fixation duration. A novel metric, the dispersion of the grouped gaze data, was also explored. Finally, a questionnaire was conducted, investigating the self-reported clarity of the different signal types.

4.1. SIGNAL TYPE

When looking at the different signal types, it clear that both message and flash relay the information of the car stopping very well. This is most evident from the reaction time data, where it could be seen that the reaction time for the off condition was on average ~ 1 s slower than both the flash and message conditions. Between the flash and message conditions, the development of the response curve is almost identical (Figure 3.3a). Even though the median average response time is ~ 0.15 s faster for the message condition compared to the flash condition, on average they are nearly identical (a difference of ~ 0.01 s). This could indicate that some participants identified the message faster than the flash, but a considerable number of participants was more hesitant for the message condition compared to the flash condition. Contradictory, the self-reported clarity revealed that the light condition was raising the alertness of the participants more than the message condition. Additionally, participants were very negative towards the off condition, while largely positive towards both the message and flash types. It has to be noted here that one participant mentioned the off condition actually raised their alertness, since they were unsure whether or not the car would stop.

From the gaze data, it follows that participants were looking at the off condition in a similar way as they were at the message condition. The flash condition was the odd one out, with a larger variance in average saccade count, a higher average saccade amplitude, and a larger spread in average fixation duration, compared to both message and off conditions. A possible explanation could be the short-lived duration of the flash message, mimicking the existing nature of a flashing signal. In comparison to the flashing signal, the message signal stayed on until the end of the trial. This could have kept the gaze of the participants locked to the signal, whereas for the flash condition, after the signal has been seen and reacted to, people could have started to scan around the scene again. This effect is noticeable as well in the dispersion data, where the dispersion increases the fastest for the flash condition after the signal has been shown. From the dispersion data it is also clear that for the off condition, participants kept their focus on the stopping car the longest, which makes sense, since they only had the car dynamics as a stopping indicator.

On average, a higher saccade count and saccade amplitude was observed for the flash condition compared to the message and off conditions. Not only did participants make more sweeps across the scene, they were also larger sweeps. A possible explanation for this could be that after the flashing signal disappears, participants are unsure where to look. They already received the signal, and may start looking for another object of interest (e.g., checking if another car is approaching).

4.2. Gaze window size

The most interesting results for the gaze window size conditions are the ones from the gaze data, as per design. The smaller the gaze window, the smaller the average number of saccades. This is contradictory to what was initially expected: an increase out of necessity to scan around for more information. The effect might be explained by the fact that the surroundings are blurred and not obscured completely. The fact that shapes could be observed, and the fact that people are not used to a gaze contingent window, might have instigated a slight reduction in saccades instead.

In line with the saccade count, the saccade amplitude shows a similar trend. Compared to the control condition without a gaze window, the small and large gaze windows appear to have a lower average saccade amplitude. The gaze contingent window might have had a suppressing effect on the ability to scan around the scene.

The fixation duration shows a near inverse of the saccade count. It appears to indicate that a smaller gaze window results in a higher fixation duration, however, the effects were not significant. This finding is also in line with the previous reasoning, where participants might have had a higher focus on the objects, resulting from the reduction in scanning around the scene.

Overall, it looks very plausible that a smaller window size caused less scanning around, and on top of that, with a smaller amplitude. In combination with the higher fixation duration, this means the eyes were less active the smaller the gaze window became. It is hard to say if this means an actual increase in cognitive load was achieved or not.

When looking at the average response times for the gaze window conditions, there is not a clear difference between the three conditions. The values between the three conditions are very close together, indicating the gaze window size had no effect on the reaction time.

4.3. Effect of gaze window size on signal type

The question that remains to be answered now is if anything can be said about the effect of the gaze window size on the message type.

It seems that the participants were quite able to keep their focus on the objects of interest, and that the gaze window did not necessarily worked distracting. In contrary, the opposite seems more likely. The gaze contingent window might have increased the focus on the object of interest. The fact that the participants scan around less with the gaze windows enabled seems to indicate that they were not looking for more information. Even though this is the case, there is no visible effect on the reaction time. The difference in reaction time between the different signal types remained similar when raising the gaze window size.

One thing that can be said about the flash type is that it does not keep the gaze locked to the car to the same degree the message type does. The fact that the focus disperses so quickly after the signal disappears indicates that participants understood the message and diverted their gaze to new potential threats in the scene. In real life this is also desirable, as it lowers the risk in getting hit by an unseen car.

The fact that all the performed n-way ANOVAs and subsequent post-hoc tests yielded no indication for significance is a strong indicator that there was, in fact, no observed effect of the gaze window size on the signal type.

4.4. HYPOTHESES AND MAIN RESEARCH QUESTION

With all these findings in mind we go over all the hypotheses we constructed at the start of this project, and attempt to formulate an answer to the main research question.

H0: Increasing the cognitive load decreases the effectiveness of the text-based eHMI There is no change in reaction time when increasing the cognitive load for text-based eHMI. There are trends visible in the between-group effects of the saccade count, and amplitude, as well as the fixation duration. There seems to be an decreased saccade count when taking a smaller gaze window, indicating participants tend to scan around the scene less. This decreased saccade count goes paired with a lowered saccade amplitude, indicating they are fewer, as well as smaller shifts of focus. The participants also tend to fixate on single objects for a longer amount of time when raising the cognitive load. These trends are not statistically significant. In short, the increase of cognitive load does not seem to decrease the effectiveness of the text-based eHMI, but the hypothesis cannot be accepted or rejected.

H0: Increasing the cognitive load decreases the effectiveness of the light-based eHMI A similar result is visible when looking at the effect of the cognitive load on the light-based eHMI. There is no change in reaction time, yet there are trends visible in the between-group effects of the saccade count, and amplitude, as well as the fixation duration. Similar to the text-based eHMI, trends are visible where the saccade count decreases with the increase of the cognitive load, paired with the same kind of decrease in saccade amplitude. Additionally, a trend is visible in the fixation duration, where the fixation duration is higher, the higher the cognitive load is. Note that these trends are not statistically significant. In short, the increase of cognitive load does not seem to decrease the effectiveness of the light-based eHMI, but the hypothesis cannot be accepted or rejected.

H0: Increasing the cognitive load decreases the effectiveness of no eHMI When looking at the effect of the cognitive load on the control group where no signal was displayed on the eHMI, it can also be noticed that there was no change in reaction time. Similar to the test conditions, a trend is visible where the saccade count decreases, and the saccade amplitude decreases, when the cognitive load is increased. Furthermore, a similar trend is visible in the fixation duration, where a trend is visible indicating an increase in fixation duration for an increased cognitive load. These trends are also not statistically significant. In short, the increase of cognitive load does not seem to decrease the effectiveness of the control group where no signal was displayed on the eHMI, but the hypothesis cannot be accepted or rejected.

H0: Increasing the cognitive load decreases the effectiveness of the text-based eHMI more than the effectiveness of light-based eHMI When comparing the effect of the increase in cognitive load on the text-based eHMI to the effect of the increase in cognitive load on the light-based eHMI, there are no clear differences. As was established in the individual parts, no change in reaction time was observed for either of the signal types when increasing the cognitive load. A slightly larger spread in the reaction times can be observed, when comparing the two conditions against each other, but there is no change. The characteristics of the gaze data are also similar. The saccade count is similar for both signal types, and both decrease by a similar amount when the cognitive load is increased. The saccade amplitude however, is slightly different. While both signal types seem to have an decrease in saccade amplitude, when increasing the cognitive load, the text-based eHMI seem to decrease slightly more, compared to its light-based counterpart. However, no statistical significance has been observed. When comparing the average fixation duration, a similar trend is observed between the two signal types. Both see a similar magnitude of increase in fixation duration when increasing the cognitive load. In conclusion, the two signal types have very similar characteristics, with respect to the increase in cognitive load. One small difference in the gaze characteristic can be observed, but no difference in the reaction times can be seen. As a result, this hypothesis cannot be rejected.

H0: Increasing the cognitive load decreases the effectiveness of no eHMI more than the effectiveness of text-based eHMI Comparing the effect of the increase in cognitive load on the effectiveness of text-based eHMI to the effect of the increase in cognitive load on the control group where no signal was shown on the eHMI, there are no clear differences. The control group had a significantly higher average reaction time compared to the text-based eHMI ($p < 0.001$ at the 95% confidence interval), but there was no statistically significant difference when taking the increase of cognitive load into account as well. The reaction times remained roughly the same when changing the cognitive load for both groups. When comparing the gaze data, no difference can be observed either. The saccade count and amplitude show proportionally similar trends for both groups when increasing the cognitive load, and the same goes for the fixation duration. This results in the conclusion that this hypothesis cannot be rejected.

H0: Increasing the cognitive load decreases the effectiveness of no eHMI more than the effectiveness of light-based eHMI Comparing the effect of the increase in cognitive load on the effectiveness of light-based eHMI to the effect of the increase in cognitive load on the control group where no signal was shown on the eHMI, there are no clear differences. When comparing the reaction time, the control group had a significantly higher average reaction time compared to the light-based eHMI ($p < 0.001$ at the 95 % confidence interval), but, similar to the text-based eHMI, there was no statistically significant difference when taking the increase of the cognitive load into account. Again, the reaction times remained similar when increasing the cognitive load. Looking at the gaze characteristics, the saccade count and amplitude show similar trend, where participants for both the light-based eHMI as well as the control group have fewer and smaller saccades on average, when increasing the cognitive load. This trend is of the same magnitude for both groups. The same can be said for the fixation duration. A proportionally similar trend is observed, where participants have longer fixation durations on average when increasing the cognitive load. Thus, it can be concluded that this hypothesis cannot be rejected.

Answering the research question Using the previous findings, an answer to the main research question can be formulated. After carefully conducting this experiment and analyzing its results, the main conclusion that has to be reached is that the cognitive load does not seem to have an influence on the effectiveness of any of the eHMI types, when compared to one another. In general, the text- and light-based eHMI show very similar response times, and feature similar gaze characteristics for the participants. Compared to the control group where no signal was shown on the eHMI, the signal clearly performs better, which is in line with previous finding.

Additional findings Other results of this experiment are in line with previous findings. The performance of the participants of both eHMI signal types outperform the control condition where no signal was shown by a large margin. The control condition had a median response time of ~ 2.23 s ($\sigma \approx 0.60$ s). Compare this to the text-based eHMI, which has a median response time of ~ 0.96 s ($\sigma \approx 0.81$ s, and $p < 0.001$ at the 95 % confidence interval), and light-based eHMI, which has a median response time of ~ 1.11 s ($\sigma \approx 0.77$ s, and $p < 0.001$ at the 95 % confidence interval).

At the same time, the response times of the text- and light-based eHMI do not differ significantly, however, when looking at the spread of the results, the light-based eHMI seems to have a slightly stronger initial response.

The dispersion also shows an interesting result. It seems to suggest that the participants use a different strategy after they made their decision. After the signal is shown, where the dispersion is at its lowest (indicating a focus on a single object for most participants), a sharp increase in the dispersion can be observed, indicating multiple participants averting their gaze from the object of interest. This is a stark contrast to the text-based eHMI, which shows a much more gradual aversion, and the control condition, where focus is maintained almost a full second longer compared to the light-based eHMI.

Multiple explanations for this observation are possible. One possible explanation lies in the way the signal was presented. The light-based eHMI featured a flashing light, that flashed twice, and then stayed off. This is similar to how traditional cars often signal pedestrians that they can cross the road safely, by flashing their head lights. This means that after the signal disappeared, the message has been relayed, and the pedestrian can look around for other objects of interest in the scene. With the text-based eHMI, the message was displayed throughout the entire braking phase, and even stayed there after coming to a complete stop. The message might have pulled the attention of the participants to the car for longer. In the case of the control condition, the fact that they stayed focused longer is most likely in the fact that there was no message displayed at all, meaning the participants did not know for sure yet if the car was stopping or not. This is also reflected in the longer reaction time, which is also ~ 1 s longer, and is in line with the maintained focus of the group of participants.

For the gaze window condition, no difference in dispersion could be observed. The participants follow a similar degree of focus throughout the trials.

When attempting to make a qualitative comparison between the light- and text-based eHMIs, the results of the self-reported clarity also hold some valuable information. Performance-wise the two types are on par, yet the self-reported clarity shows a slightly different picture. While the self-reported scores are similar in terms of usefulness of the system, they differ slightly in satisfaction. The text-based eHMI is slightly preferred among most of the participants, outscoring its light-based counterpart on most of the satisfying components, having many participants indicate that this type was more pleasant and more desirable than the light-based eHMI.

It has to be seen if text-based eHMI truly can become reality, since cultural differences and lawmakers also play a huge role in the potential realization of such a system.

4.5. LIMITATIONS AND RECOMMENDATIONS

One of the biggest limitations in this research has been the relatively small sample size. Only 23 participants were able to participate during the span of the testing phase. Ideally this number would be a lot higher, but time constraints simply did not allow for more. The results that were found can be used as preliminary findings. Since some trends were visible in the results, a high enough sample size could turn those trends into statistical significance. The question remains if this is desirable, since none of the visible trends actually pointed towards answering the main research question.

Following from this, a second limitation can be identified. Since no clear effect was noticeable in the effectiveness of the signals, in terms of reaction time, when raising the cognitive load, it raises the following question: *Was the cognitive load raised by enough, to make useful statements?* While the gaze characteristics suggest the cognitive load was definitely raised, the lack of any effect begs this question. The possibility exists that the peripheral blur was not strong enough.

Along this line of thought, different types of gaze contingent windows could also be tested. Some ideas include: make the peripheral black, make the peripheral darkened, increase the blur even further, use even smaller gaze windows. It would benefit future research to investigate the effect of different types of gaze windows, including stronger blur, before attempting to investigate its effect on the signal type.

Something that could be added to make sure the cognitive load is better measurable, is to include a questionnaire at the end, querying the participants about the perceived cognitive load. When including the user experience to the equation, a better understanding of the results can be reached. This is something that could have benefited this research as well, as it was hard to indicate if the added gaze window did, in fact, increase the cognitive load.

Something else that was initially planned to be included in the analysis of the gaze data is the size of the pupils over time, since they could be a strong indicator for increase in cognitive load. However, this recorded data varied greatly, even between eyes of the same participant. So the decision was made to leave this data out of consideration. An average pupil size of both eyes might still yield useful results, but for the scope of this research, the focus instead was turned to the gaze characteristics (i.e., saccade count, amplitude and fixation duration).

Another possible limitation, mentioned earlier in the conclusion, lies in the fact that the different signal types were not visible for the same amount of time. The light-based signal was two short flashes, after which the eHMI stayed off for the remaining duration of the trial, while the text-based signal would show the message until the end of the trial. This means that in theory, the light-based signal has a higher chance to be missed by the participant. This is not reflected in the results, but when doing future research, this is something that is undesirable. Ideally the time each signal is shown to the participants should be identical.

A final thought is that due to way the experiment was set up, participants might have reached the conclusion that whenever a signal was shown, the car showing that signal would always stop. This type of learning effect was attempted to be avoided, by including trials where no car would stop at all, but by not including trials where cars show a signal indicating they would not stop, a learning effect may still be present for the cases a signal was shown.

Bibliography

[1] ambientCG. *Ground 036*. ambientCG. Dec. 28, 2019. URL: https://ambientcg.com/view?id=Ground036 (visited on 10/20/2021).

[2] ambientCG. *Paving Stones 038*. ambientCG. Mar. 13, 2019. URL: https://ambientcg.com/view?id=PavingStones038 (visited on 12/02/2020).

[3] AMD. *AMD Radeon™ ProRender*. 2017. URL: https://www.amd.com/en/technologies/radeon-prorender/.

[4] Audi. *Audi A8 - Audi AI traffic jam pilot - Audi Technology Portal*. 2017. URL: https://www.audi-technology-portal.de/en/electrics-electronics/driver-assistant-systems/audi-a8-audi-ai-traffic-jam-pilot (visited on 11/25/2019).

[5] Marc Barendse. *The Effects of Information Type on Pedestrian Crossing Decisions*. MA thesis. TU Delft, 2019.

[6] Pavlo Bazilinskyy, Dimitra Dodou, and J. C. F. De Winter. "Survey on eHMI concepts: The effect of text, color, and perspective". In: *Transportation Research Part F: Traffic Psychology and Behaviour* 67 (Nov. 2019), pp. 175–194. DOI: 10.1016/j.trf.2019.10.013.

[7] Marc-Philipp Böckle et al. "SAV2P – Exploring the Impact of an Interface for Shared Automated Vehicles on Pedestrians' Experience". In: *Proceedings of the 9th International Conference on Automotive User Interfaces and Interactive Vehicular Applications Adjunct*. New York, New York, USA: ACM, Sept. 2017, pp. 136–140. ISBN: 9781450351515. DOI: 10.1145/3131726.3131765. URL: http://dl.acm.org/citation.cfm?doid=3131726.3131765.

[8] C.C. Studio. *Tesla Model S (3D model)*. ArtStation. Aug. 23, 2019. URL: https://ccstudio30.artstation.com/store/GwVA/tesla-model-s-3d-model (visited on 12/11/2019).

[9] Dylan Calluy. *Tesla in Antwerp*. Jan. 23, 2020. URL: https://unsplash.com/photos/D1w-heh9uMg (visited on 06/30/2022).

[10] Carlos Cardona. *Low Poly Nature Set for Blender*. Gumroad. Apr. 27, 2021. URL: https://chuckcg.gumroad.com/l/joGTC (visited on 10/20/2021).

[11] Melissa Cefkin et al. "Multi-methods Research to Examine External HMI for Highly Automated Vehicles". In: *HCI in Mobility, Transport, and Automotive Systems*. Ed. by Heidi Krömker. Vol. 11596. Lecture Notes in Computer Science. Springer International Publishing, 2019, pp. 46–64. ISBN: 978-3-030-22665-7. DOI: 10.1007/978-3-030-22666-4_4. URL: http://link.springer.com/10.1007/978-3-030-22666-4%7B%5C_%7D4.

[12] Koen de Clercq et al. "External Human-Machine Interfaces on Automated Vehicles: Effects on Pedestrian Crossing Decisions". In: *Human Factors: The Journal of the Human Factors and Ergonomics Society* 61.8 (Mar. 2019), pp. 1353–1370. ISSN: 15478181. DOI: 10.1177/0018720819836343.

[13] Chandler Creech et al. "Trust and Control in Autonomous Vehicle Interactions". In: *Workshop on "Morality and Social Trust in Autonomoy", Robotics Science and Systems Conference*. 2017. URL: https://deepblue.lib.umich.edu/handle/2027.42/137661.

[14] Felipe Del Rio. *Procedural Buildings with Geometry Nodes*. Gumroad. June 3, 2021. URL: https://felipedelrio.gumroad.com/l/msJMr (visited on 10/20/2021).

[15] Y. B. Eisma et al. "External Human-Machine Interfaces: The Effect of Display Location on Crossing Intentions and Eye Movements". In: *Information* 11.October (2019), p. 13. DOI: 10.3390/info11010013.

[16] Adrian Etter and Marc Biedermann. *Edf2Mat© Matlab Toolbox*. Version 1.20. University of Zurich, Department of Economics, Zurich, May 22, 2018. URL: https://github.com/uzh/edf-converter.

[17] Stafanie M. Faas and Martin Baumann. "Yielding Light Signal Evaluation for Self-driving Vehicle and Pedestrian Interaction". In: *Human Systems Engineering and Design II*. Springer International Publishing, Aug. 2019, pp. 189–194. DOI: 10.1007/978-3-030-27928-8_29.

[18] Lex Fridman et al. “To Walk or Not to Walk: Crowdsourced Assessment of External Vehicle-to-Pedestrian Displays”. In: (July 10, 2017). arXiv: 1707.02698 [cs.HC]. URL: http://arxiv.org/abs/1707.02698.

[19] Hui-Ting Goh, Miranda Pearce, and Asha Vas. “Task matters: an investigation on the effect of different secondary tasks on dual-task gait in older adults”. In: *BMC Geriatrics* 21.1 (Sept. 2021). DOI: 10.1186/s12877-021-02464-8.

[20] Ann-Christin Hensch et al. “How Should Automated Vehicles Communicate? – Effects of a Light-Based Communication Approach in a Wizard-of-Oz Study”. In: *Advances in Human Factors of Transportation*. Springer International Publishing, June 2019, pp. 79–91. ISBN: 9783030205027. DOI: 10.1007/978-3-030-20503-4_8. URL: http://link.springer.com/10.1007/978-3-030-20503-4_8.

[21] Intel®. *Intel® Open Image Denoise*. 2018. URL: https://www.openimagedenoise.org/.

[22] Aditya Jain et al. “A comparative study of visual and auditory reaction times on the basis of gender and physical activity levels of medical first year students”. In: *International Journal of Applied and Basic Medical Research* 5.2 (2015), p. 124. DOI: 10.4103/2229-516x.157168.

[23] Hiroaki Kazui, Hajime Kitagaki, and Etsuro Mari. “Cortical activation during retrieval of arithmetical facts and actual calculation: A functional magnetic resonance imaging study”. In: *Psychiatry and Clinical Neurosciences* 54.4 (Aug. 2000), pp. 479–485. DOI: 10.1046/j.1440-1819.2000.00739.x.

[24] Sangseok Lee and Dong Kyu Lee. “What is the proper way to apply the multiple comparison test?” In: *Korean Journal of Anesthesiology* 71.5 (Oct. 2018), pp. 353–360. DOI: 10.4097/kja.d.18.00242.

[25] D. Welty Lefever. “Measuring Geographic Concentration by Means of the Standard Deviational Ellipse”. In: *American Journal of Sociology* 32.1 (July 1926), pp. 88–94. URL: https://www.jstor.org/stable/2765249.

[26] Rensis Likert. “A Technique for the Measurement of Attitudes”. In: *Archives of Psychology* 140 (1932), pp. 1–55.

[27] Jacques Lucke and Omar Emara. *Animation Nodes*. Version 2.1.4. Mar. 6, 2019. URL: https://animation-nodes.com/.

[28] LuxCoreRender project. *LuxCoreRender*. Version 2.x. 2017. URL: https://luxcorerender.org/.

[29] Sergej Majboroda. *Autumn Park HDRI - Poly Haven*. Dec. 3, 2019. URL: https://polyhaven.com/a/autumn_park (visited on 12/02/2020).

[30] Ministerie van Verkeer en Waterstaat. *Reglement verkeersregels en verkeerstekens*. 1990. URL: https://wetten.overheid.nl/BWBR0004825/ (visited on 10/22/2021).

[31] OTOY. *OctaneRender®*. 2012. URL: https://home.otoy.com/render/octane-render/.

[32] Pixar. *RenderMan*. 2015. URL: https://renderman.pixar.com/.

[33] Kazumi Renge. “Effect of driving experience on drivers’ decoding process of roadway interpersonal communication”. In: *Ergonomics* 43.1 (Jan. 2000), pp. 27–39. ISSN: 0014-0139. DOI: 10.1080/001401300184648. URL: https://www.tandfonline.com/doi/full/10.1080/001401300184648.

[34] SAE. *SAE J3016™ Recommended Practice: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. Tech. rep. SAE, 2021.

[35] Anna Schieben et al. “Designing the interaction of automated vehicles with other traffic participants: design considerations based on human needs and expectations”. In: *Cognition, Technology And Work* 21.1 (Sept. 2018), pp. 69–85. ISSN: 1435-5558. DOI: 10.1007/s10111-018-0521-z. URL: http://link.springer.com/10.1007/s10111-018-0521-z.

[36] Henri Schmidt et al. “Hacking Nonverbal Communication Between Pedestrians and Vehicles in Virtual Reality”. In: (Apr. 2, 2019). arXiv: 1904.01931 [cs.HC]. URL: http://arxiv.org/abs/1904.01931.

[37] Ye Eun Song et al. “External HMIs and Their Effect on the Interaction Between Pedestrians and Automated Vehicles”. In: *Intelligent Human Systems Integration*. Vol. 722. Springer International Publishing, Dec. 2017, pp. 13–18. ISBN: 9783319738871. DOI: 10.1007/978-3-319-73888-8_3. URL: http://link.springer.com/10.1007/978-3-319-73888-8_3.

[38] SR Research Ltd. *Circular Gaze Contingent Window*. Mississauga, Ontario, Canada, Nov. 5, 2020. URL: https://www.sr-support.com/thread-7307.html (visited on 01/18/2022).

[39] SR Research Ltd. *EDF Data Structure*. Mississauga, Ontario, Canada, Aug. 27, 2020. URL: https://www.sr-support.com/thread-54.html (visited on 03/23/2022).

[40] SR Research Ltd. *EyeLink® 1000 Plus*. Mississauga, Ontario, Canada, 2017. URL: https://www.sr-research.com/eyelink-1000-plus/.

[41] SR Research Ltd. *EyeLink® 1000 Plus User Manual*. Version 1.0.18. Mississauga, Ontario, Canada, 2021.

[42] SR Research Ltd. *SR Research Experiment Builder*. Version 2.3.38. Mississauga, Ontario, Canada, Aug. 26, 2020.

[43] SR Research Ltd. *SR Research Experiment Builder User Manual*. Version 2.3.1. Mississauga, Ontario, Canada, 2020.

[44] Masashige Suwa, Masatoshi Nishimura, and Reiko Sakata. "LED Projection Module enables a vehicle to communicate with pedestrians and other vehicles". In: *2017 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, 2017, pp. 37–38. ISBN: 978-1-5090-5544-9. DOI: 10.1109/icce.2017.7889220. URL: http://ieeexplore.ieee.org/document/7889220/.

[45] The appleseedhq Organization. *appleseed*. 2010. URL: https://appleseedhq.net/.

[46] The Blender Foundation. *Blender*. Version 2.93.5. Oct. 5, 2021. URL: https://www.blender.org/.

[47] The Blender Foundation. *Cycles*. 2019. URL: https://www.cycles-renderer.org/.

[48] The Blender Foundation. *Eevee*. 2018. URL: https://docs.blender.org/manual/en/latest/render/eevee/index.html (visited on 2020).

[49] The FFmpeg Developers. *FFmpeg*. Version 4.4.1. 2021. URL: https://ffmpeg.org/.

[50] The MathWorks, Inc. *MATLAB*. Version R2022a. Natick, Massachusetts, 2022. URL: https://www.mathworks.com/products/matlab.html.

[51] Jinke D. Van der Laan, Adriaan Heino, and Dick De Waard. *A simple procedure for the assessment of acceptance of advanced transport telematics*. Feb. 1997. DOI: 10.1016/s0968-090x(96)00025-3.

[52] Bin Wang, Wenzhong Shi, and Zelang Miao. "Confidence Analysis of Standard Deviational Ellipse and Its Extension into Higher Dimensional Euclidean Space". In: *PLOS ONE* 10.3 (Mar. 2015). Ed. by Duccio Rocchini, e0118537. DOI: 10.1371/journal.pone.0118537.

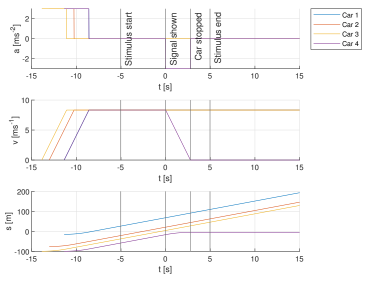
[53] YafaRay Team. *YafaRay*. Version 3.x. 2016. URL: http://yafaray.org.

[54] Moses Yoo. *spider plot*. Version 17.8. Mar. 24, 2022. URL: https://github.com/NewGuy012/spider_plot (visited on 04/25/2022).

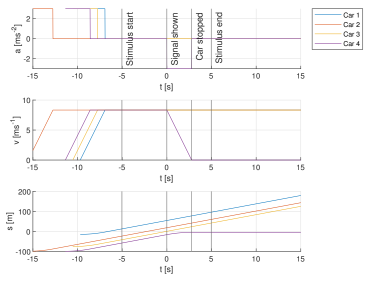
[55] Robert S. Yuill. "The Standard Deviational Ellipse; An Updated Tool for Spatial Description". In: *Geografiska Annaler: Series B, Human Geography* 53.1 (Apr. 1971), pp. 28–39. DOI: 10.1080/04353684.1971.11879353. URL: https://www.jstor.org/stable/490885.

A
Experiment

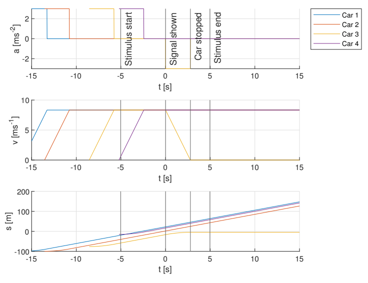
A.1. Car profiles



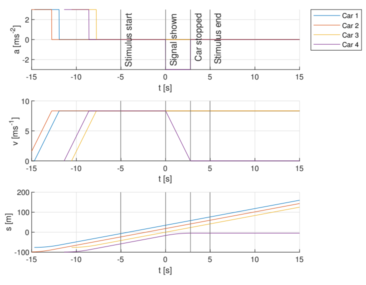
(a) Car profiles used in preset 1.



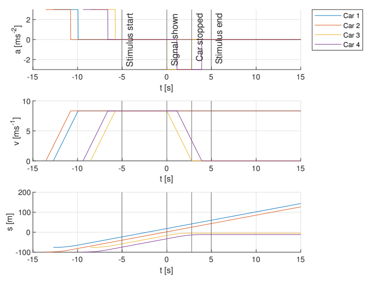
(b) Car profiles used in preset 2.



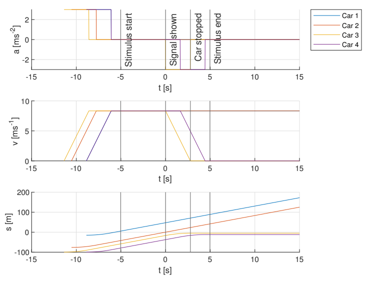
(c) Car profiles used in preset 3.



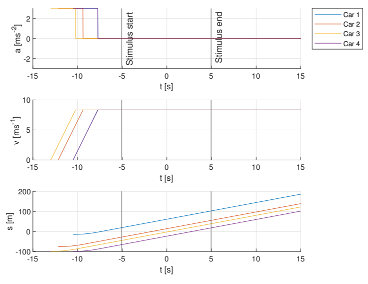
(d) Car profiles used in preset 4.



(e) Car profiles used in preset 5.



(f) Car profiles used in preset 6.



(g) Car profiles used in preset 7.

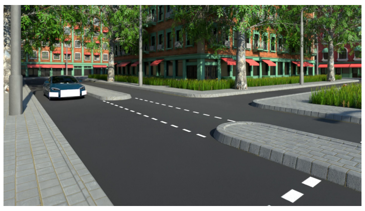
Figure A.1: Calculated acceleration (a), velocity (v), and travelled distance (s) profiles per `Car` object. The travelled distances are normalized around the participant's position ($s > 0$ indicates a car has passed the participant).

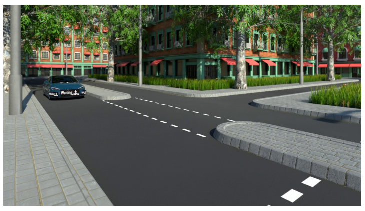
A.2. PRESET KEYFRAMES

(a) Stimulus start, flash (frame 1).

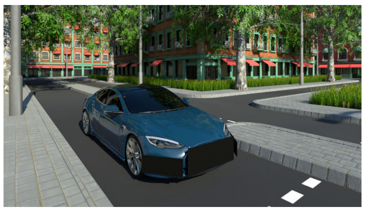
(b) Stimulus start, message (frame 1).

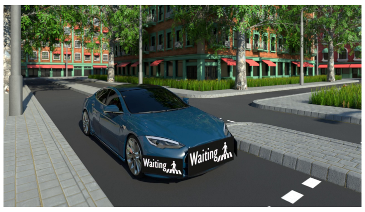
(c) Stimulus start, off (frame 1).


(d) Signal shown, flash (frame 302).


(e) Signal shown, message (frame 302).

(f) Signal shown, off (frame 302).

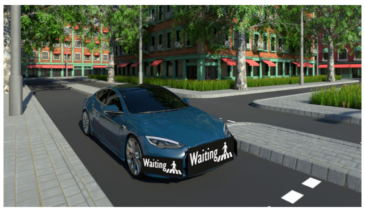

(g) Car stopped, flash (frame 469).



(h) Car stopped, message (frame 469).

(i) Car stopped, off (frame 469).

(j) Stimulus end, flash (frame 600).



(k) Stimulus end, message (frame 600).

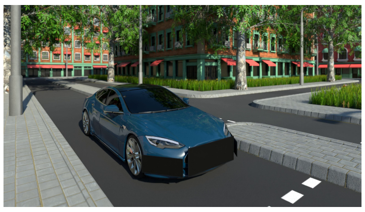

(l) Stimulus end, off (frame 600).

Figure A.2: Screenshots of preset 1.

(a) Stimulus start, flash (frame 1).

(b) Stimulus start, message (frame 1).

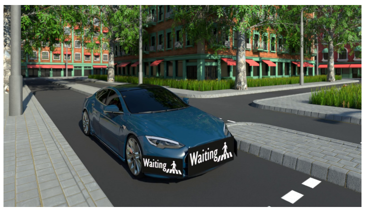
(c) Stimulus start, off (frame 1).

(d) Signal shown, flash (frame 302).

(e) Signal shown, message (frame 302).

(f) Signal shown, off (frame 302).

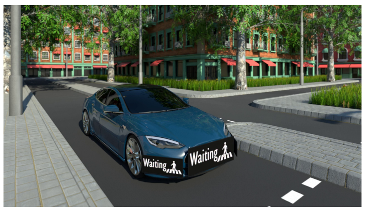
(g) Car stopped, flash (frame 469).



(h) Car stopped, message (frame 469).

(i) Car stopped, off (frame 469).

(j) Stimulus end, flash (frame 600).



(k) Stimulus end, message (frame 600).

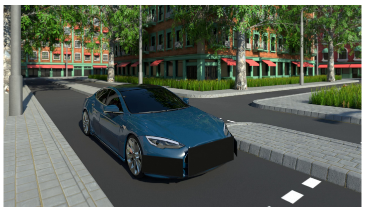

(l) Stimulus end, off (frame 600).

Figure A.3: Screenshots of preset 2.

(a) Stimulus start, flash (frame 1).

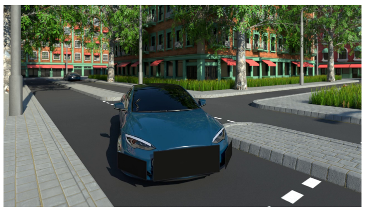
(b) Stimulus start, message (frame 1).

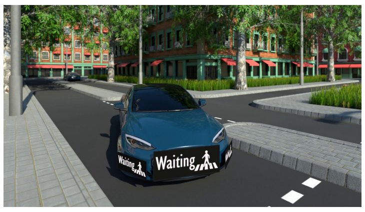
(c) Stimulus start, off (frame 1).

(d) Signal shown, flash (frame 302).

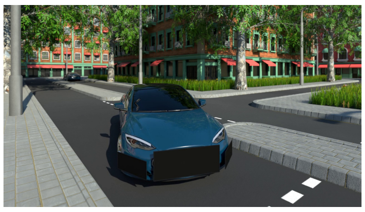
(e) Signal shown, message (frame 302).

(f) Signal shown, off (frame 302).

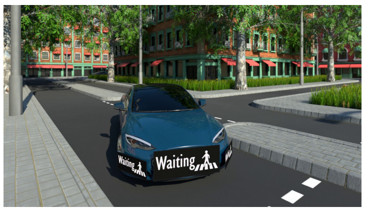

(g) Car stopped, flash (frame 469).



(h) Car stopped, message (frame 469).


(i) Car stopped, off (frame 469).

(j) Stimulus end, flash (frame 600).



(k) Stimulus end, message (frame 600).

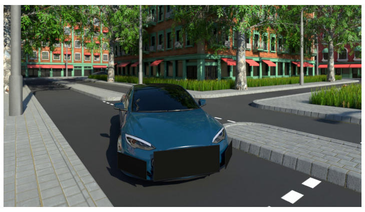
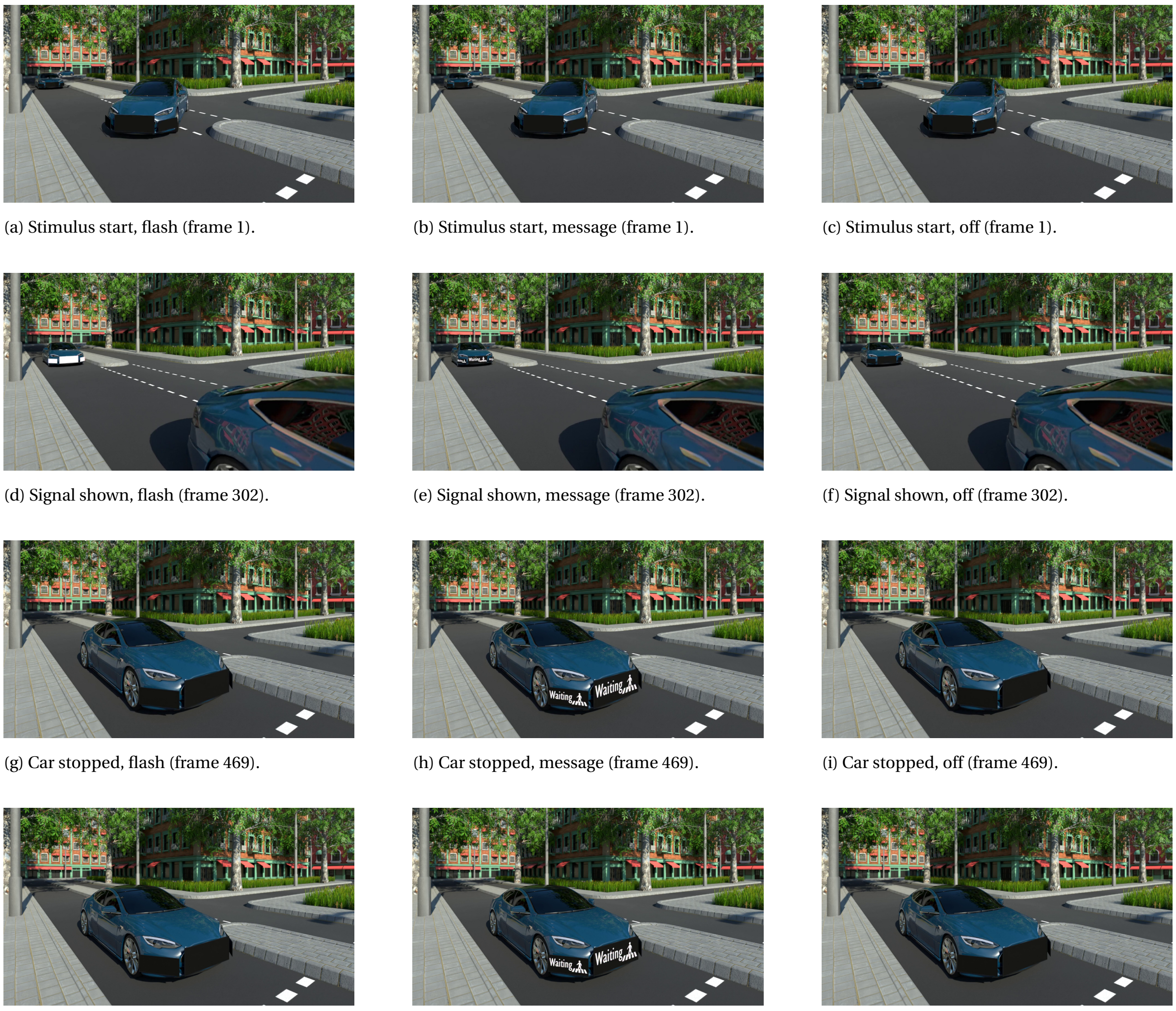

(l) Stimulus end, off (frame 600).

Figure A.4: Screenshots of preset 3.



(a) Stimulus start, flash (frame 1).

(b) Stimulus start, message (frame 1).

(c) Stimulus start, off (frame 1).

(d) Signal shown, flash (frame 302).

(e) Signal shown, message (frame 302).

(f) Signal shown, off (frame 302).

(g) Car stopped, flash (frame 469).

(h) Car stopped, message (frame 469).

(i) Car stopped, off (frame 469).

(j) Stimulus end, flash (frame 600).

(k) Stimulus end, message (frame 600).

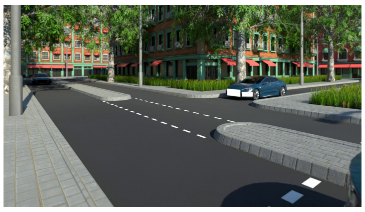
(l) Stimulus end, off (frame 600).

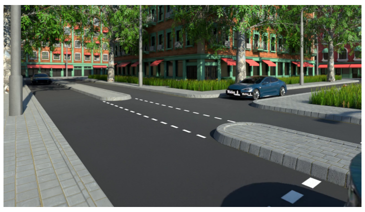
Figure A.5: Screenshots of preset 4.

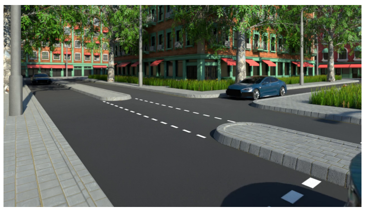
(a) Stimulus start, flash (frame 1).

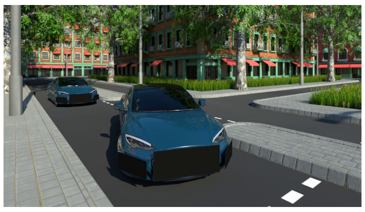
(b) Stimulus start, message (frame 1).

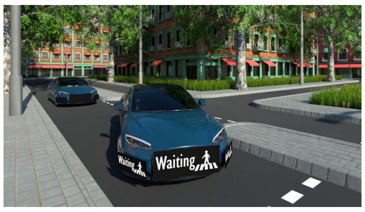
(c) Stimulus start, off (frame 1).


(d) Signal shown, flash (frame 302).

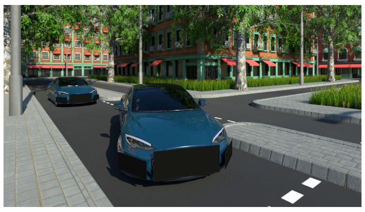

(e) Signal shown, message (frame 302).

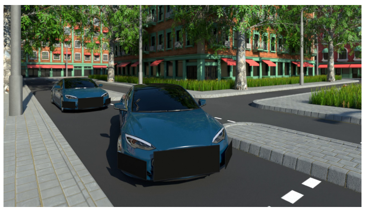

(f) Signal shown, off (frame 302).

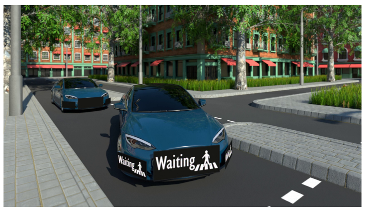

(g) Car stopped, flash (frame 469).



(h) Car stopped, message (frame 469).


(i) Car stopped, off (frame 469).


(j) Stimulus end, flash (frame 600).



(k) Stimulus end, message (frame 600).

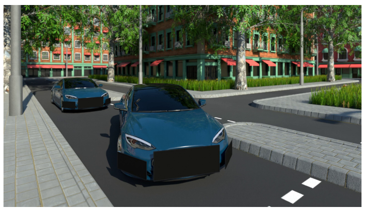
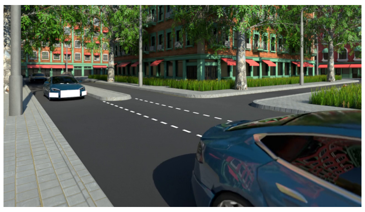

(l) Stimulus end, off (frame 600).

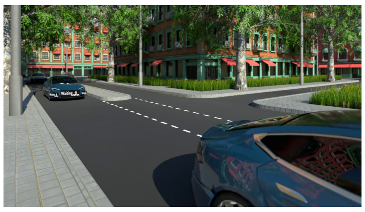
Figure A.6: Screenshots of preset 5.

(a) Stimulus start, flash (frame 1).

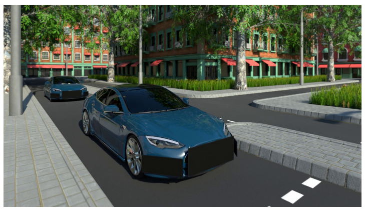
(b) Stimulus start, message (frame 1).

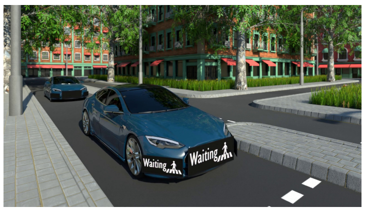
(c) Stimulus start, off (frame 1).


(d) Signal shown, flash (frame 302).


(e) Signal shown, message (frame 302).

(f) Signal shown, off (frame 302).

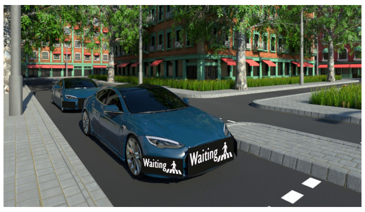

(g) Car stopped, flash (frame 469).



(h) Car stopped, message (frame 469).

(i) Car stopped, off (frame 469).

(j) Stimulus end, flash (frame 600).



(k) Stimulus end, message (frame 600).

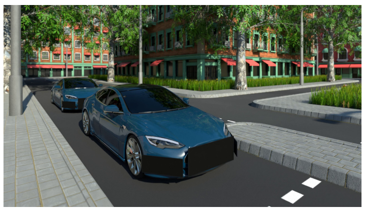

(l) Stimulus end, off (frame 600).

Figure A.7: Screenshots of preset 6.

(a) Stimulus start, flash (frame 1).

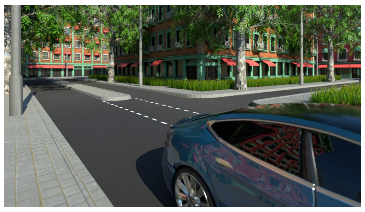
(b) Stimulus start, message (frame 1).

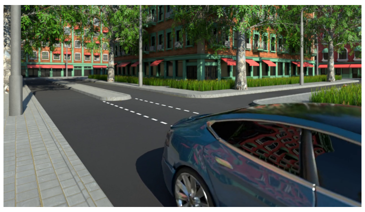
(c) Stimulus start, off (frame 1).

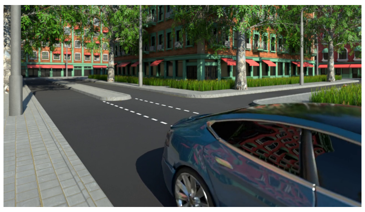
(d) Nothing shown, flash (frame 302).

(e) Nothing shown, message (frame 302).

(f) Nothing shown, off (frame 302).


(g) Not stopped, flash (frame 469).


(h) Not stopped, message (frame 469).


(i) Not stopped, off (frame 469).

(j) Stimulus end, flash (frame 600).

(k) Stimulus end, message (frame 600).

(l) Stimulus end, off (frame 600).

Figure A.8: Screenshots of preset 7.

A.3. TRIAL TABLE

Table A.1: Overview of the trials, including the different conditions.

Trial	Preset	Signal type	Gaze window size
1	1	FLASH	SMALL
2	1	MESSAGE	SMALL
3	1	OFF	SMALL
4	2	FLASH	SMALL
5	2	MESSAGE	SMALL
6	2	OFF	SMALL
7	3	FLASH	SMALL
8	3	MESSAGE	SMALL
9	3	OFF	SMALL
10	4	FLASH	SMALL
11	4	MESSAGE	SMALL
12	4	OFF	SMALL
13	5	FLASH	SMALL
14	5	MESSAGE	SMALL
15	5	OFF	SMALL
16	6	FLASH	SMALL
17	6	MESSAGE	SMALL
18	6	OFF	SMALL
19	7	FLASH	SMALL
20	7	MESSAGE	SMALL
21	7	OFF	SMALL
22	1	FLASH	LARGE
23	1	MESSAGE	LARGE
24	1	OFF	LARGE
25	2	FLASH	LARGE
26	2	MESSAGE	LARGE
27	2	OFF	LARGE
28	3	FLASH	LARGE
29	3	MESSAGE	LARGE
30	3	OFF	LARGE
31	4	FLASH	LARGE
32	4	MESSAGE	LARGE
33	4	OFF	LARGE
34	5	FLASH	LARGE
35	5	MESSAGE	LARGE
36	5	OFF	LARGE
37	6	FLASH	LARGE
38	6	MESSAGE	LARGE
39	6	OFF	LARGE
40	7	FLASH	LARGE
41	7	MESSAGE	LARGE
42	7	OFF	LARGE
43	1	FLASH	NONE
44	1	MESSAGE	NONE
45	1	OFF	NONE
46	2	FLASH	NONE
47	2	MESSAGE	NONE
48	2	OFF	NONE
49	3	FLASH	NONE
50	3	MESSAGE	NONE
51	3	OFF	NONE
52	4	FLASH	NONE
53	4	MESSAGE	NONE
54	4	OFF	NONE
55	5	FLASH	NONE
56	5	MESSAGE	NONE
57	5	OFF	NONE
58	6	FLASH	NONE
59	6	MESSAGE	NONE
60	6	OFF	NONE
61	7	FLASH	NONE
62	7	MESSAGE	NONE
63	7	OFF	NONE

B
Results

B.1. 1-way ANOVA summary tables

Table B.1: Summary of the average response times (s) results per signal type condition.

Condition	mean	sum	min	max	range	median	σ^2	σ	n
FLASH	1.33	30.54	0.47	3.73	3.25	1.11	0.60	0.77	23
MESSAGE	1.34	30.90	0.42	3.66	3.23	0.96	0.66	0.81	23
OFF	2.31	53.06	1.23	4.17	2.94	2.23	0.36	0.60	23

Table B.2: Summary of the average response times (s) results per gaze window condition.

Condition	mean	sum	min	max	range	median	σ^2	σ	n
SMALL	1.66	38.28	0.68	3.60	2.93	1.50	0.44	0.66	23
LARGE	1.72	39.55	0.70	4.03	3.32	1.47	0.58	0.76	23
NONE	1.60	36.73	0.75	3.89	3.15	1.33	0.49	0.70	23

Table B.3: Summary of the average saccade counts results per signal type condition.

Condition	mean	sum	min	max	range	median	σ^2	σ	n
FLASH	17.69	406.83	9.56	25.50	15.94	17.72	16.77	4.10	23
MESSAGE	18.52	426.06	12.44	26.50	14.06	18.28	15.83	3.98	23
OFF	17.63	405.44	9.67	25.17	15.50	18.11	15.20	3.90	23

Table B.4: Summary of the average saccade counts results per gaze window condition.

Condition	mean	sum	min	max	range	median	σ^2	σ	n
SMALL	16.50	379.56	9.83	25.06	15.22	16.83	13.11	3.62	23
LARGE	17.92	412.17	10.28	25.72	15.44	18.11	14.94	3.87	23
NONE	19.42	446.61	10.94	29.06	18.11	19.61	21.32	4.62	23

Table B.5: Summary of the average saccade amplitudes (px) results per signal type condition.

Condition	mean	sum	min	max	range	median	σ^2	σ	n
FLASH	229.26	5273.06	119.67	352.19	232.52	222.33	2756.90	52.51	23
MESSAGE	212.60	4889.70	113.43	344.98	231.56	201.78	2707.71	52.04	23
OFF	219.12	5039.67	133.01	354.35	221.34	213.48	2599.35	50.98	23

Table B.6: Summary of the average saccade amplitudes (px) results per gaze window condition.

Condition	mean	sum	min	max	range	median	σ^2	σ	n
SMALL	210.65	4844.97	107.27	351.87	244.59	197.94	3049.36	55.22	23
LARGE	215.43	4954.78	121.53	331.55	210.02	205.81	2416.33	49.16	23
NONE	234.90	5402.67	137.30	368.10	230.80	233.76	2899.66	53.85	23

Table B.7: Summary of the average fixation durations (ms) results per signal type condition.

Condition	mean	sum	min	max	range	median	σ^2	σ	n
FLASH	605.73	13931.85	398.32	869.09	470.77	578.77	21698.48	147.30	23
MESSAGE	573.31	13186.18	377.99	806.48	428.49	566.78	14402.81	120.01	23
OFF	598.75	13771.15	405.39	837.03	431.64	561.36	16405.15	128.08	23

Table B.8: Summary of the average fixation durations (ms) results per gaze window condition.

Condition	mean	sum	min	max	range	median	σ^2	σ	n
SMALL	636.80	14646.41	405.27	885.22	479.96	599.10	17388.60	131.87	23
LARGE	583.66	13424.13	388.97	799.17	410.19	568.57	15147.21	123.07	23
NONE	557.33	12818.65	348.09	905.54	557.45	508.85	21396.30	146.27	23

B.2. 1-WAY ANOVA TABLES

Table B.9: ANOVA performed on the average response times (s) results between signal type conditions.

Source	Sum Sq.	df	Mean Sq.	F	Prob>F
Groups	14.47	2	7.23	13.42	1.29×10^{-5}
Error	35.58	66	0.54		
Total	50.05	68			

Table B.10: ANOVA performed on the average response times (s) results between gaze window conditions.

Source	Sum Sq.	df	Mean Sq.	F	Prob>F
Groups	0.17	2	0.09	0.17	0.84
Error	33.32	66	0.50		
Total	33.49	68			

Table B.11: ANOVA performed on the average saccade counts results between signal type conditions.

Source	Sum Sq.	df	Mean Sq.	F	Prob>F
Groups	11.54	2	5.77	0.36	0.70
Error	1051.62	66	15.93		
Total	1063.16	68			

Table B.12: ANOVA performed on the average saccade counts results between gaze window conditions.

Source	Sum Sq.	df	Mean Sq.	F	Prob>F
Groups	97.77	2	48.89	2.97	0.06
Error	1086.04	66	16.46		
Total	1183.81	68			

Table B.13: ANOVA performed on the average saccade amplitudes (px) results between signal type conditions.

Source	Sum Sq.	df	Mean Sq.	F	Prob>F
Groups	3245.33	2	1622.66	0.60	0.55
Error	177 407.11	66	2687.99		
Total	180 652.44	68			

Table B.14: ANOVA performed on the average saccade amplitudes (px) results between gaze window conditions.

Source	Sum Sq.	df	Mean Sq.	F	Prob>F
Groups	7589.75	2	3794.87	1.36	0.26
Error	184 037.66	66	2788.45		
Total	191 627.40	68			

Table B.15: ANOVA performed on the average fixation durations (s) results between signal type conditions.

Source	Sum Sq.	df	Mean Sq.	F	Prob>F
Groups	13 391.87	2	6695.93	0.38	0.68
Error	1 155 141.67	66	17 502.15		
Total	1 168 533.53	68			

Table B.16: ANOVA performed on the average fixation durations (s) results between gaze window conditions.

Source	Sum Sq.	df	Mean Sq.	F	Prob>F
Groups	75 381.22	2	37 690.61	2.10	0.13
Error	1 186 506.45	66	17 977.37		
Total	1 261 887.67	68			

B.3. 1-WAY ANOVA POST-HOC COMPARISON TABLES

Table B.17: Multcompare performed on the results of the ANOVA of the average response times (s) between signal type conditions, applying the Bonferroni method.

Conditions	diff	ci(min)	ci(max)	*p*
FLASH-MESSAGE	−0.02	−0.55	0.52	1.000
FLASH-OFF	−0.98	−1.51	−0.45	< 0.001
MESSAGE-OFF	−0.96	−1.50	−0.43	< 0.001

Table B.18: Multcompare performed on the results of the ANOVA of the average response times (s) between gaze window conditions, applying the Bonferroni method.

Conditions	diff	ci(min)	ci(max)	*p*
SMALL-LARGE	−0.06	−0.57	0.46	1.000
SMALL-NONE	0.07	−0.45	0.58	1.000
LARGE-NONE	0.12	−0.39	0.64	1.000

Table B.19: Multcompare performed on the results of the ANOVA of the average saccade counts between signal type conditions, applying the Bonferroni method.

Conditions	diff	ci(min)	ci(max)	*p*
FLASH-MESSAGE	−0.84	−3.73	2.06	1.000
FLASH-OFF	0.06	−2.83	2.95	1.000
MESSAGE-OFF	0.90	−2.00	3.79	1.000

Table B.20: Multcompare performed on the results of the ANOVA of the average saccade counts between gaze window conditions, applying the Bonferroni method.

Conditions	diff	ci(min)	ci(max)	*p*
SMALL-LARGE	−1.42	−4.36	1.52	0.720
SMALL-NONE	−2.92	−5.85	0.02	0.053
LARGE-NONE	−1.50	−4.44	1.44	0.645

Table B.21: Multcompare performed on the results of the ANOVA of the average saccade amplitudes (px) between signal type conditions, applying the Bonferroni method.

Conditions	diff	ci(min)	ci(max)	p
FLASH-MESSAGE	16.67	−20.89	54.22	0.839
FLASH-OFF	10.15	−27.41	47.70	1.000
MESSAGE-OFF	−6.52	−44.08	31.04	1.000

Table B.22: Multcompare performed on the results of the ANOVA of the average saccade amplitudes (px) between gaze window conditions, applying the Bonferroni method.

Conditions	diff	ci(min)	ci(max)	p
SMALL-LARGE	−4.77	−43.03	33.48	1.000
SMALL-NONE	−24.25	−62.50	14.00	0.373
LARGE-NONE	−19.47	−57.73	18.78	0.647

Table B.23: Multcompare performed on the results of the ANOVA of the average fixation durations (s) between signal type conditions, applying the Bonferroni method.

Conditions	diff	ci(min)	ci(max)	p
FLASH-MESSAGE	32.42	−63.41	128.25	1.000
FLASH-OFF	6.99	−88.85	102.82	1.000
MESSAGE-OFF	−25.43	−121.27	70.40	1.000

Table B.24: Multcompare performed on the results of the ANOVA of the average fixation durations (s) between gaze window conditions, applying the Bonferroni method.

Conditions	diff	ci(min)	ci(max)	p
SMALL-LARGE	53.14	−43.98	150.27	0.551
SMALL-NONE	79.47	−17.66	176.59	0.146
LARGE-NONE	26.33	−70.80	123.45	1.000

B.4. N-WAY ANOVA SUMMARY TABLES

Table B.25: Summary of the average response times (s) results per signal type condition.

Condition	mean	sum	min	max	range	median	σ^2	σ	n
FLASH-SMALL	1.29	29.77	0.42	2.89	2.47	1.13	0.40	0.63	23
FLASH-LARGE	1.35	31.11	0.37	3.94	3.57	1.12	0.75	0.87	23
FLASH-NONE	1.32	30.41	0.53	4.19	3.66	1.09	0.72	0.85	23
MESSAGE-SMALL	1.36	31.32	0.38	3.73	3.35	1.00	0.76	0.87	23
MESSAGE-LARGE	1.40	32.22	0.37	3.82	3.44	1.09	0.75	0.86	23
MESSAGE-NONE	1.27	29.15	0.51	3.41	2.90	0.98	0.56	0.75	23
OFF-SMALL	2.33	53.58	1.23	4.07	2.84	2.17	0.38	0.61	23
OFF-LARGE	2.40	55.30	1.34	4.59	3.25	2.41	0.44	0.67	23
OFF-NONE	2.20	50.55	1.09	4.05	2.96	2.08	0.38	0.62	23

Table B.26: Summary of the average response times (s) results per signal type condition.

Condition	mean	sum	min	max	range	median	σ^2	σ	n
FLASH-SMALL	16.36	376.17	8.67	23.17	14.50	16.17	14.97	3.87	23
FLASH-LARGE	17.98	413.50	8.17	25.67	17.50	18.50	18.79	4.34	23
FLASH-NONE	18.73	430.83	9.83	27.67	17.83	18.50	21.82	4.67	23
MESSAGE-SMALL	16.94	389.67	11.17	28.50	17.33	16.50	15.47	3.93	23
MESSAGE-LARGE	18.08	415.83	12.17	27.00	14.83	17.17	19.01	4.36	23
MESSAGE-NONE	20.55	472.67	12.00	31.00	19.00	20.00	22.66	4.76	23
OFF-SMALL	16.21	372.83	9.50	24.17	14.67	16.83	14.94	3.87	23
OFF-LARGE	17.70	407.17	10.50	24.50	14.00	18.00	13.45	3.67	23
OFF-NONE	18.97	436.33	9.00	28.50	19.50	18.67	24.63	4.96	23

Table B.27: Summary of the average response times (s) results per signal type condition.

Condition	mean	sum	min	max	range	median	σ^2	σ	n
FLASH-SMALL	218.20	5018.50	108.88	344.42	235.54	212.82	3206.79	56.63	23
FLASH-LARGE	220.76	5077.40	133.42	341.43	208.01	216.85	3082.25	55.52	23
FLASH-NONE	248.84	5723.27	116.70	378.30	261.61	248.09	3272.26	57.20	23
MESSAGE-SMALL	201.38	4631.74	103.18	345.39	242.20	185.05	3680.92	60.67	23
MESSAGE-LARGE	214.38	4930.73	114.04	336.25	222.21	214.17	2489.80	49.90	23
MESSAGE-NONE	222.03	5106.62	123.06	353.32	230.25	210.02	2937.25	54.20	23
OFF-SMALL	212.38	4884.68	109.76	373.38	263.63	203.36	3172.76	56.33	23
OFF-LARGE	211.14	4856.20	117.14	316.99	199.85	203.95	2502.96	50.03	23
OFF-NONE	233.83	5378.13	117.48	372.69	255.21	240.97	3305.84	57.50	23

Table B.28: Summary of the average response times (s) results per signal type condition.

Condition	mean	sum	min	max	range	median	σ^2	σ	n
FLASH-SMALL	646.86	14877.71	434.82	945.29	510.47	622.76	24370.14	156.11	23
FLASH-LARGE	582.33	13393.68	395.53	856.10	460.57	580.01	17414.65	131.96	23
FLASH-NONE	588.01	13524.16	363.14	1007.10	643.96	514.55	36008.89	189.76	23
MESSAGE-SMALL	614.92	14143.19	348.08	832.02	483.95	600.39	15635.72	125.04	23
MESSAGE-LARGE	587.44	13511.01	369.20	955.19	585.98	599.17	22064.51	148.54	23
MESSAGE-NONE	517.58	11904.34	325.79	849.18	523.39	499.70	15038.50	122.63	23
OFF-SMALL	648.62	14918.33	432.90	967.46	534.56	627.94	20182.26	142.06	23
OFF-LARGE	581.20	13367.70	386.50	854.97	468.46	574.53	15996.33	126.48	23
OFF-NONE	566.41	13027.44	352.23	909.54	557.31	541.86	22358.33	149.53	23

B.5. N-WAY ANOVA TABLES

Table B.29: ANOVA performed on the average response times (s) results between signal type conditions.

Source	Sum Sq.	df	Singular	Mean Sq.	F	Prob>F
X1	43.95	2	0	21.98	38.48	7.65×10^{-15}
X2	0.53	2	0	0.26	0.46	0.63
X1*X2	0.23	4	0	0.06	0.10	0.98
Error	113.09	198	0	0.57		
Total	157.80	206	0			

Table B.30: ANOVA performed on the average response times (s) results between signal type conditions.

Source	Sum Sq.	df	Singular	Mean Sq.	F	Prob>F
X1	34.62	2	0	17.31	0.94	0.39
X2	293.32	2	0	146.66	7.96	0.00
X1*X2	18.97	4	0	4.74	0.26	0.90
Error	3646.28	198	0	18.42		
Total	3993.19	206	0			

Table B.31: ANOVA performed on the average response times (s) results between signal type conditions.

Source	Sum Sq.	df	Singular	Mean Sq.	F	Prob>F
X1	9735.98	2	0	4867.99	1.58	0.21
X2	22 769.24	2	0	11 384.62	3.71	0.03
X1*X2	3025.97	4	0	756.49	0.25	0.91
Error	608 318.02	198	0	3072.31		
Total	643 849.21	206	0			

Table B.32: ANOVA performed on the average response times (s) results between signal type conditions.

Source	Sum Sq.	df	Singular	Mean Sq.	F	Prob>F
X1	40 175.60	2	0	20 087.80	0.96	0.39
X2	226 143.66	2	0	113 071.83	5.38	0.01
X1*X2	36 764.71	4	0	9191.18	0.44	0.78
Error	4 159 525.49	198	0	21 007.70		
Total	4 462 609.46	206	0			

B.6. N-WAY ANOVA POST-HOC COMPARISON TABLES

Table B.33: Multcompare performed on the results of the ANOVA of the average response times (s) between signal type conditions, applying the Bonferroni method.

Conditions	diff	ci(min)	ci(max)	*p*
FLASH+SMALL-MESSAGE+SMALL	−0.07	−0.79	0.66	1.000
FLASH+SMALL-OFF+SMALL	−1.04	−1.76	−0.31	< 0.001
FLASH+SMALL-FLASH+LARGE	−0.06	−0.78	0.66	1.000
FLASH+SMALL-MESSAGE+LARGE	−0.11	−0.83	0.62	1.000
FLASH+SMALL-OFF+LARGE	−1.11	−1.83	−0.39	< 0.001
FLASH+SMALL-FLASH+NONE	−0.03	−0.75	0.69	1.000
FLASH+SMALL-MESSAGE+NONE	0.03	−0.70	0.75	1.000
FLASH+SMALL-OFF+NONE	−0.90	−1.63	−0.18	0.003
MESSAGE+SMALL-OFF+SMALL	−0.97	−1.69	−0.25	< 0.001
MESSAGE+SMALL-FLASH+LARGE	0.01	−0.71	0.73	1.000
MESSAGE+SMALL-MESSAGE+LARGE	−0.04	−0.76	0.68	1.000
MESSAGE+SMALL-OFF+LARGE	−1.04	−1.76	−0.32	< 0.001
MESSAGE+SMALL-FLASH+NONE	0.04	−0.68	0.76	1.000
MESSAGE+SMALL-MESSAGE+NONE	0.09	−0.63	0.82	1.000
MESSAGE+SMALL-OFF+NONE	−0.84	−1.56	−0.11	0.008
OFF+SMALL-FLASH+LARGE	0.98	0.25	1.70	< 0.001
OFF+SMALL-MESSAGE+LARGE	0.93	0.21	1.65	0.002
OFF+SMALL-OFF+LARGE	−0.07	−0.80	0.65	1.000
OFF+SMALL-FLASH+NONE	1.01	0.28	1.73	< 0.001
OFF+SMALL-MESSAGE+NONE	1.06	0.34	1.79	< 0.001
OFF+SMALL-OFF+NONE	0.13	−0.59	0.85	1.000
FLASH+LARGE-MESSAGE+LARGE	−0.05	−0.77	0.67	1.000
FLASH+LARGE-OFF+LARGE	−1.05	−1.77	−0.33	< 0.001
FLASH+LARGE-FLASH+NONE	0.03	−0.69	0.75	1.000
FLASH+LARGE-MESSAGE+NONE	0.09	−0.64	0.81	1.000
FLASH+LARGE-OFF+NONE	−0.85	−1.57	−0.12	0.007
MESSAGE+LARGE-OFF+LARGE	−1.00	−1.73	−0.28	< 0.001
MESSAGE+LARGE-FLASH+NONE	0.08	−0.64	0.80	1.000
MESSAGE+LARGE-MESSAGE+NONE	0.13	−0.59	0.86	1.000
MESSAGE+LARGE-OFF+NONE	−0.80	−1.52	−0.07	0.016
OFF+LARGE-FLASH+NONE	1.08	0.36	1.80	< 0.001
OFF+LARGE-MESSAGE+NONE	1.14	0.41	1.86	< 0.001
OFF+LARGE-OFF+NONE	0.21	−0.52	0.93	1.000
FLASH+NONE-MESSAGE+NONE	0.06	−0.67	0.78	1.000
FLASH+NONE-OFF+NONE	−0.88	−1.60	−0.15	0.004
MESSAGE+NONE-OFF+NONE	−0.93	−1.65	−0.21	0.002

Table B.34: Multcompare performed on the results of the ANOVA of the average response times (s) between signal type conditions, applying the Bonferroni method.

Conditions	diff	ci(min)	ci(max)	p
FLASH+SMALL-MESSAGE+SMALL	−0.59	−4.69	3.52	1.000
FLASH+SMALL-OFF+SMALL	0.14	−3.96	4.25	1.000
FLASH+SMALL-FLASH+LARGE	−1.62	−5.73	2.48	1.000
FLASH+SMALL-MESSAGE+LARGE	−1.72	−5.83	2.38	1.000
FLASH+SMALL-OFF+LARGE	−1.35	−5.45	2.76	1.000
FLASH+SMALL-FLASH+NONE	−2.38	−6.48	1.73	1.000
FLASH+SMALL-MESSAGE+NONE	−4.20	−8.30	−0.09	0.039
FLASH+SMALL-OFF+NONE	−2.62	−6.72	1.49	1.000
MESSAGE+SMALL-OFF+SMALL	0.73	−3.37	4.84	1.000
MESSAGE+SMALL-FLASH+LARGE	−1.04	−5.14	3.07	1.000
MESSAGE+SMALL-MESSAGE+LARGE	−1.14	−5.24	2.97	1.000
MESSAGE+SMALL-OFF+LARGE	−0.76	−4.86	3.34	1.000
MESSAGE+SMALL-FLASH+NONE	−1.79	−5.89	2.31	1.000
MESSAGE+SMALL-MESSAGE+NONE	−3.61	−7.71	0.49	0.173
MESSAGE+SMALL-OFF+NONE	−2.03	−6.13	2.07	1.000
OFF+SMALL-FLASH+LARGE	−1.77	−5.87	2.34	1.000
OFF+SMALL-MESSAGE+LARGE	−1.87	−5.97	2.23	1.000
OFF+SMALL-OFF+LARGE	−1.49	−5.60	2.61	1.000
OFF+SMALL-FLASH+NONE	−2.52	−6.63	1.58	1.000
OFF+SMALL-MESSAGE+NONE	−4.34	−8.44	−0.24	0.026
OFF+SMALL-OFF+NONE	−2.76	−6.86	1.34	1.000
FLASH+LARGE-MESSAGE+LARGE	−0.10	−4.21	4.00	1.000
FLASH+LARGE-OFF+LARGE	0.28	−3.83	4.38	1.000
FLASH+LARGE-FLASH+NONE	−0.75	−4.86	3.35	1.000
FLASH+LARGE-MESSAGE+NONE	−2.57	−6.68	1.53	1.000
FLASH+LARGE-OFF+NONE	−0.99	−5.10	3.11	1.000
MESSAGE+LARGE-OFF+LARGE	0.38	−3.73	4.48	1.000
MESSAGE+LARGE-FLASH+NONE	−0.65	−4.76	3.45	1.000
MESSAGE+LARGE-MESSAGE+NONE	−2.47	−6.57	1.63	1.000
MESSAGE+LARGE-OFF+NONE	−0.89	−4.99	3.21	1.000
OFF+LARGE-FLASH+NONE	−1.03	−5.13	3.07	1.000
OFF+LARGE-MESSAGE+NONE	−2.85	−6.95	1.26	0.919
OFF+LARGE-OFF+NONE	−1.27	−5.37	2.84	1.000
FLASH+NONE-MESSAGE+NONE	−1.82	−5.92	2.28	1.000
FLASH+NONE-OFF+NONE	−0.24	−4.34	3.86	1.000
MESSAGE+NONE-OFF+NONE	1.58	−2.52	5.68	1.000

Table B.35: Multcompare performed on the results of the ANOVA of the average response times (s) between signal type conditions, applying the Bonferroni method.

Conditions	diff	ci(min)	ci(max)	*p*
FLASH+SMALL-MESSAGE+SMALL	16.82	−36.19	69.82	1.000
FLASH+SMALL-OFF+SMALL	5.82	−47.19	58.82	1.000
FLASH+SMALL-FLASH+LARGE	−2.56	−55.56	50.44	1.000
FLASH+SMALL-MESSAGE+LARGE	3.82	−49.19	56.82	1.000
FLASH+SMALL-OFF+LARGE	7.06	−45.95	60.06	1.000
FLASH+SMALL-FLASH+NONE	−30.64	−83.65	22.36	1.000
FLASH+SMALL-MESSAGE+NONE	−3.83	−56.83	49.17	1.000
FLASH+SMALL-OFF+NONE	−15.64	−68.64	37.37	1.000
MESSAGE+SMALL-OFF+SMALL	−11.00	−64.00	42.01	1.000
MESSAGE+SMALL-FLASH+LARGE	−19.38	−72.38	33.63	1.000
MESSAGE+SMALL-MESSAGE+LARGE	−13.00	−66.00	40.00	1.000
MESSAGE+SMALL-OFF+LARGE	−9.76	−62.76	43.24	1.000
MESSAGE+SMALL-FLASH+NONE	−47.46	−100.46	5.55	0.148
MESSAGE+SMALL-MESSAGE+NONE	−20.65	−73.65	32.36	1.000
MESSAGE+SMALL-OFF+NONE	−32.45	−85.46	20.55	1.000
OFF+SMALL-FLASH+LARGE	−8.38	−61.38	44.62	1.000
OFF+SMALL-MESSAGE+LARGE	−2.00	−55.01	51.00	1.000
OFF+SMALL-OFF+LARGE	1.24	−51.77	54.24	1.000
OFF+SMALL-FLASH+NONE	−36.46	−89.46	16.54	0.966
OFF+SMALL-MESSAGE+NONE	−9.65	−62.65	43.35	1.000
OFF+SMALL-OFF+NONE	−21.45	−74.46	31.55	1.000
FLASH+LARGE-MESSAGE+LARGE	6.38	−46.63	59.38	1.000
FLASH+LARGE-OFF+LARGE	9.62	−43.39	62.62	1.000
FLASH+LARGE-FLASH+NONE	−28.08	−81.09	24.92	1.000
FLASH+LARGE-MESSAGE+NONE	−1.27	−54.27	51.73	1.000
FLASH+LARGE-OFF+NONE	−13.08	−66.08	39.93	1.000
MESSAGE+LARGE-OFF+LARGE	3.24	−49.76	56.24	1.000
MESSAGE+LARGE-FLASH+NONE	−34.46	−87.46	18.55	1.000
MESSAGE+LARGE-MESSAGE+NONE	−7.65	−60.65	45.36	1.000
MESSAGE+LARGE-OFF+NONE	−19.45	−72.46	33.55	1.000
OFF+LARGE-FLASH+NONE	−37.70	−90.70	15.31	0.796
OFF+LARGE-MESSAGE+NONE	−10.89	−63.89	42.12	1.000
OFF+LARGE-OFF+NONE	−22.69	−75.70	30.31	1.000
FLASH+NONE-MESSAGE+NONE	26.81	−26.19	79.82	1.000
FLASH+NONE-OFF+NONE	15.01	−38.00	68.01	1.000
MESSAGE+NONE-OFF+NONE	−11.80	−64.81	41.20	1.000

Table B.36: Multcompare performed on the results of the ANOVA of the average response times (s) between signal type conditions, applying the Bonferroni method.

Conditions	diff	ci(min)	ci(max)	p
FLASH+SMALL-MESSAGE+SMALL	31.94	−106.67	170.54	1.000
FLASH+SMALL-OFF+SMALL	−1.77	−140.37	136.83	1.000
FLASH+SMALL-FLASH+LARGE	64.52	−74.08	203.12	1.000
FLASH+SMALL-MESSAGE+LARGE	59.42	−79.18	198.02	1.000
FLASH+SMALL-OFF+LARGE	65.65	−72.95	204.25	1.000
FLASH+SMALL-FLASH+NONE	58.85	−79.75	197.45	1.000
FLASH+SMALL-MESSAGE+NONE	129.28	−9.32	267.88	0.101
FLASH+SMALL-OFF+NONE	80.45	−58.15	219.05	1.000
MESSAGE+SMALL-OFF+SMALL	−33.70	−172.30	104.90	1.000
MESSAGE+SMALL-FLASH+LARGE	32.59	−106.01	171.19	1.000
MESSAGE+SMALL-MESSAGE+LARGE	27.49	−111.11	166.09	1.000
MESSAGE+SMALL-OFF+LARGE	33.72	−104.88	172.32	1.000
MESSAGE+SMALL-FLASH+NONE	26.91	−111.69	165.52	1.000
MESSAGE+SMALL-MESSAGE+NONE	97.34	−41.26	235.94	0.858
MESSAGE+SMALL-OFF+NONE	48.51	−90.09	187.11	1.000
OFF+SMALL-FLASH+LARGE	66.29	−72.31	204.89	1.000
OFF+SMALL-MESSAGE+LARGE	61.19	−77.41	199.79	1.000
OFF+SMALL-OFF+LARGE	67.42	−71.18	206.02	1.000
OFF+SMALL-FLASH+NONE	60.62	−77.98	199.22	1.000
OFF+SMALL-MESSAGE+NONE	131.04	−7.56	269.64	0.089
OFF+SMALL-OFF+NONE	82.21	−56.39	220.81	1.000
FLASH+LARGE-MESSAGE+LARGE	−5.10	−143.70	133.50	1.000
FLASH+LARGE-OFF+LARGE	1.13	−137.47	139.73	1.000
FLASH+LARGE-FLASH+NONE	−5.67	−144.27	132.93	1.000
FLASH+LARGE-MESSAGE+NONE	64.75	−73.85	203.35	1.000
FLASH+LARGE-OFF+NONE	15.92	−122.68	154.52	1.000
MESSAGE+LARGE-OFF+LARGE	6.23	−132.37	144.83	1.000
MESSAGE+LARGE-FLASH+NONE	−0.57	−139.17	138.03	1.000
MESSAGE+LARGE-MESSAGE+NONE	69.86	−68.75	208.46	1.000
MESSAGE+LARGE-OFF+NONE	21.02	−117.58	159.63	1.000
OFF+LARGE-FLASH+NONE	−6.80	−145.40	131.80	1.000
OFF+LARGE-MESSAGE+NONE	63.62	−74.98	202.22	1.000
OFF+LARGE-OFF+NONE	14.79	−123.81	153.39	1.000
FLASH+NONE-MESSAGE+NONE	70.43	−68.17	209.03	1.000
FLASH+NONE-OFF+NONE	21.60	−117.00	160.20	1.000
MESSAGE+NONE-OFF+NONE	−48.83	−187.43	89.77	1.000

B.7. HEATMAPS

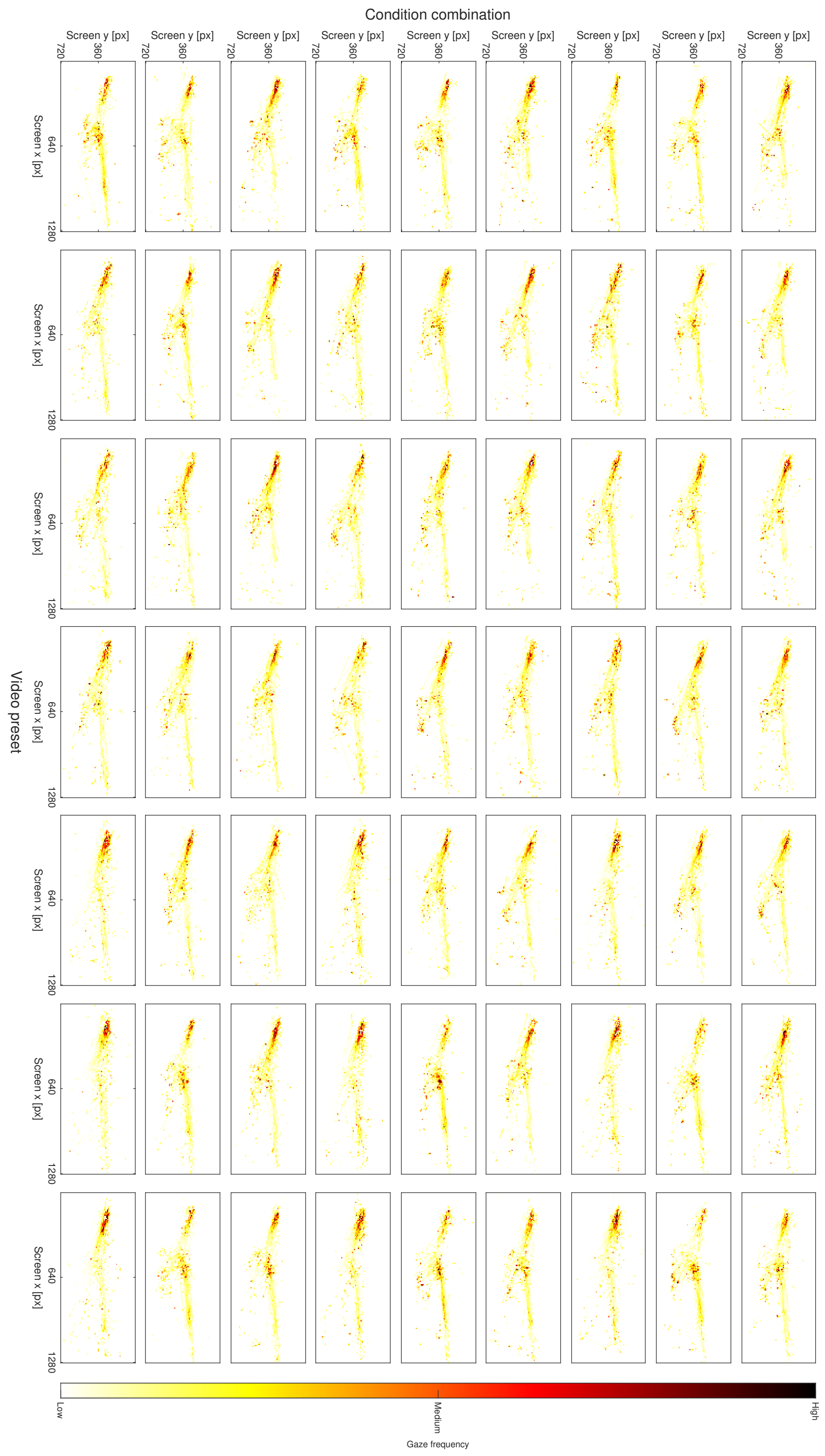


Figure B.1: Overview of the heat maps. Each row represents a combination of conditions. Each column represents a trial preset.

C
CODE LISTINGS

C.1. MATLAB

C.1.1. MAIN

```
%% Main
% Initial code based on the code published with:
% - https://www.mdpi.com/2078-2489/11/1/13
% Expanded and refactored to fit this experiment's needs.
% The code was created with MATLAB R2022a.
% The code uses functions from the following toolboxes:
% - Computer Vision Toolbox 10.2
% - Signal Processing Toolbox 9.0
% - Statistics and Machine Learning Toolbox 12.3
clear
clc

timerMain = tic;

% Initialize script variables.
run Main_Init.m

% Import .edf's and RESULTS_FILE.txt's.
run Main_Import.m

% Prepare .edf and RESULTS_FILE.txt data.
run Main_PrepareData.m

% Perform statistical tests.
run Main_Statistics.m

% Create various graphs.
run Main_Plot.m

% Draw gaze on stimuli and reconstruct trials.
run Main_Redraw.m

% Finalize
disp(strcat("Total elapsed time is ",string(toc(timerMain))," seconds."))
```

Listing 2: `Main.m`

```
%% Main_Init
% Initialize script variables.
```

```
resultsRoot = '..\..\..\..\GazeEHMI_deployed\results';
edfList = dir(fullfile(resultsRoot, '**\*.edf'));
edfList = edfList(2:end); % ignore test session
txtList = dir(fullfile(resultsRoot,'**\RESULTS_FILE.txt'));
txtList = txtList(2:end); % ignore test session
dataFile = 'EyeTrackingData.mat';
trialTable = readtable('TrialNumberIdentifiers.csv');
figureDir = '.\figures';
statsDir = '.\stats';

% Set ordinal orders and resort trialTable accordingly. This will affect
% the order of the conditions in the graphs.
trialTable.EHMI = categorical(trialTable.EHMI, ...
    {'FLASH','MESSAGE','OFF'},"Ordinal",true);
trialTable.MASK = categorical(trialTable.MASK, ...
    {'SMALL','LARGE','NONE'},"Ordinal",true);
trialTable = sortrows(trialTable, ...
    [trialTable.Properties.VariableNames(3), ...
    trialTable.Properties.VariableNames(1), ...
    trialTable.Properties.VariableNames(2)]);

% Eyetracker and stimulus details.
constants.SAMPLERATE = 2000; % [Hz]
constants.FRAMERATE = 60; % [fps]
constants.PADDEDDURATION = 1; % [s]
constants.DURATION = 10; % [s]

% Participant and trial details.
constants.NUMPARTICIPANTS = 23; % [-]
constants.NUMTESTTRIALS = 3; % [-]
constants.NUMTRIALS = 63; % [-]

% Screen resolution used for recording.
constants.SCREENWIDTH = 1280; % [px]
constants.SCREENHEIGHT = 720; % [px]

% Exact frame number the signal was shown to participants: frame 302.
% Exact number of frames after the signal frame the car has come to a full
% stop: 167 frames.
constants.SIGNALFRAME = 302; % [-]
constants.STOPPEDFRAME = constants.SIGNALFRAME + 167; % [-]

% Event values.
constants.KEYPRESSED = 1;
constants.KEYRELEASED = -1;

% Minimum reaction time.
% (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4456887/)
constants.MINREACTIONTIME = 0.18; % [s]

% Trial conditions.
conditions.EHMI = categorical(categories(trialTable.EHMI), ...
    categories(trialTable.EHMI),"Ordinal",true);
conditions.MASK = categorical(categories(trialTable.MASK), ...
    categories(trialTable.MASK),"Ordinal",true);

% Set several time frames, shifted such that the signal is shown at t=0.
sampleSpan = linspace(1/constants.SAMPLERATE,constants.DURATION, ...
    constants.SAMPLERATE*constants.DURATION) ...
    - constants.SIGNALFRAME/constants.FRAMERATE;
timeSpan = linspace(1/constants.FRAMERATE,constants.DURATION, ...
    constants.FRAMERATE*constants.DURATION) ...
    - constants.SIGNALFRAME/constants.FRAMERATE;

% Scale factor, used to get 95% confidence interval on 2d spatial data.
% (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4358977/)
scaleFactor = 2.4477;

% Get current screen details.
set(0,'units','pixels')
screenSize = get(0,"ScreenSize");
screenSize = [screenSize(3), screenSize(4)];

% Set figure size.
constants.FIGURESIZEX = 560;
constants.FIGURESIZEY = 420;
figureSize = [constants.FIGURESIZEX, constants.FIGURESIZEY];

% Import switch.
importFromFiles = true;
```

Listing 3: `Main_Init.m`

```
%% Main_Import
% Import .edf's and RESULTS_FILE.txt's.
% Importing from source files takes approx. 6 minutes, importing from .mat
% file takes approx. 5 seconds.
if importFromFiles == true
    disp("Importing and extracting .edf data.")
    [data.raw.time, ...
            data.raw.gazeX, ...
            data.raw.gazeY, ...
            data.raw.pupilSize, ...
            trialOrder] = ...
        ImportFromEdfs( ...
            edfList, ...
            trialTable, ...
            constants);

    disp("Importing and extracting RESULTS_FILE.txt data.")
    data.raw.keyPresses = ...
        ImportFromTxts( ...
            txtList, ...
            trialTable, ...
            constants);

    disp(strcat("Writing extracted data to ",dataFile,"."))
    tic
    save(dataFile,'data','trialOrder','-v7.3')
    toc
else
    disp(strcat("Importing extracted .edf and RESULTS_FILE.txt data "+ ...
        "from ",dataFile,"."))
    tic
    load(dataFile)
    toc
end
```

Listing 4: `Main_Import.m`

```
%% Main_PrepareData
% Process raw .edf data.
[data.processed.time, ...
        data.processed.gazeX, ...
        data.processed.gazeY, ...
        data.processed.pupilSize, ...
        data.metrics.fixations, ...
        data.metrics.saccades] = ...
    PrepareEdfData( ...
        data.raw.time, ...
        data.raw.gazeX, ...
        data.raw.gazeY, ...
        data.raw.pupilSize, ...
        trialOrder, ...
        constants);

% Process raw .txt data.
data.metrics.keyPresses = ...
    PrepareTxtData( ...
        data.raw.keyPresses, ...
        trialTable, ...
        constants);

% Get accumulated keypress tables.
data.metrics.keyPressesCumulative = ...
    AccumulateKeyPresses( ...
        data.metrics.keyPresses, ...
        constants);
data.metrics.keyPressesCumulativeCondition = ...
    AccumulateKeyPressesConditions( ...
        data.metrics.keyPresses, ...
        trialTable, ...
        conditions, ...
        constants);

% Get accumulated saccade tables.
data.metrics.saccadesCumulative = ...
    AccumulateSaccades( ...
        data.metrics.saccades, ...
        constants);
data.metrics.saccadesCumulativeCondition = ...
    AccumulateSaccadesConditions( ...
        data.metrics.saccades, ...
        trialTable, ...
        conditions, ...
        constants);

% Calculate response times.
data.metrics.responseTime = ...
    GetResponseTime( ...
        data.metrics.keyPresses, ...
        constants);

% Prepare heat maps.
gridsize = 8;
heatMapsSeparate = zeros( ...
    constants.SCREENHEIGHT/gridsize, ...
    constants.SCREENWIDTH/gridsize, ...
    constants.NUMTRIALS);

```

```
% Create heat maps for each trial.
for trial = 1:constants.NUMTRIALS
    heatMapsSeparate(:,:,trial) = PrepareHeatMap( ...
        cat(1,data.processed.gazeX{trial,:}), ...
        cat(1,data.processed.gazeY{trial,:}), ...
        gridsize, ...
        constants);
end

% Clean artifacts and scale result.
heatMapsSeparate(heatMapsSeparate>2000) = 0;
heatMapsSeparate = NormalizeMatrix(heatMapsSeparate);

% Create heat maps for totals.
gridsize = 1;
heatMapTotalRaw = PrepareHeatMap( ...
    cat(1,data.raw.gazeX{:}), ...
    cat(1,data.raw.gazeY{:}), ...
    gridsize, ...
    constants);

heatMapTotalProcessed = PrepareHeatMap( ...
    cat(1,data.processed.gazeX{:}), ...
    cat(1,data.processed.gazeY{:}), ...
    gridsize, ...
    constants);

% Clean artifacts and scale result.
heatMapTotalRaw = NormalizeMatrix(heatMapTotalRaw);
heatMapTotalProcessed(heatMapTotalProcessed>2000) = 0;
heatMapTotalProcessed = NormalizeMatrix(heatMapTotalProcessed);

% Calculate dispersion.
[data.metrics.meanX, ...
        data.metrics.meanY, ...
        data.metrics.stdXPrime, ...
        data.metrics.stdYPrime, ...
        data.metrics.theta] = ...
    CalculateDispersion( ...
        data.processed.gazeX, ...
        data.processed.gazeY, ...
        constants);

% Calculate eccentricity.
data.metrics.eccentricity = ...
    CalculateEccentricity( ...
        data.metrics.stdXPrime, ...
        data.metrics.stdYPrime);

% Calculate ellipse area that encompasses the 95% interval. Calculate with
% sigma, then scale with scaleFactor^2 to get the 95% interval. See
% Main_Init.m for more information.
data.metrics.ellipseArea = ...
    scaleFactor^2*CalculateEllipseArea( ...
        data.metrics.stdXPrime, ...
        data.metrics.stdYPrime);

% Theoretical maximum 2*sigma ellipse area size.
data.metrics.stdXMax = sqrt((constants.SCREENWIDTH/2)^2/2);
data.metrics.stdYMax = sqrt((constants.SCREENHEIGHT/2)^2/2);
data.metrics.ellipseMaxArea = ...
```

```
    scaleFactor^2*CalculateEllipseArea( ...
        data.metrics.stdXMax, ...
        data.metrics.stdYMax);

% Calculate eccentricity and ellipse area means.
[data.metrics.eccentricityMean, ...
        data.metrics.ellipseMeanArea] = ...
    PrepareEllipseData( ...
        data.metrics.eccentricity, ...
        data.metrics.ellipseArea, ...
        trialTable, ...
        conditions, ...
        constants);

% Calculate averages per condition type for saccade and fixation data.
[data.metrics.ehmiResponseTimes, ...
        data.metrics.maskResponseTimes, ...
        data.metrics.ehmiSaccadeCounts, ...
        data.metrics.maskSaccadeCounts, ...
        data.metrics.ehmiSaccadeAverageAmplitudes, ...
        data.metrics.maskSaccadeAverageAmplitudes, ...
        data.metrics.ehmiFixationAverageDurations, ...
        data.metrics.maskFixationAverageDurations] = ...
    PrepareBoxChartData(data.metrics.responseTime, ...
        data.metrics.saccades, ...
        data.metrics.fixations, ...
        trialTable, ...
        conditions, ...
        constants);

% Set categorical group arrays for boxcharts and scatter plots.
ehmiGroup = repmat(categorical(conditions.EHMI), ...
    constants.NUMPARTICIPANTS,1);
maskGroup = repmat(categorical(conditions.MASK), ...
    constants.NUMPARTICIPANTS,1);

data.metrics.ehmiResponseTimeMeans = ...
    groupsummary(data.metrics.ehmiResponseTimes, ...
    ehmiGroup,"mean");
data.metrics.maskResponseTimeMeans = ...
    groupsummary(data.metrics.maskResponseTimes, ...
    maskGroup,"mean");
data.metrics.ehmiSaccadeCountsMeans = ...
    groupsummary(data.metrics.ehmiSaccadeCounts, ...
    ehmiGroup,"mean");
data.metrics.maskSaccadeCountsMeans = ...
    groupsummary(data.metrics.maskSaccadeCounts, ...
    maskGroup,"mean");
data.metrics.ehmiSaccadeAverageAmplitudesMeans = ...
    groupsummary(data.metrics.ehmiSaccadeAverageAmplitudes, ...
    ehmiGroup,"mean");
data.metrics.maskSaccadeAverageAmplitudesMeans = ...
    groupsummary(data.metrics.maskSaccadeAverageAmplitudes, ...
    maskGroup,"mean");
data.metrics.ehmiFixationAverageDurationsMeans = ...
    groupsummary(data.metrics.ehmiFixationAverageDurations, ...
    ehmiGroup,"mean");
data.metrics.maskFixationAverageDurationsMeans = ...
    groupsummary(data.metrics.maskFixationAverageDurations, ...
    maskGroup,"mean");

```

```matlab
% Calculate averages per condition type 1 for each condition type 2.
[data.metrics.responseTimeMeans, ...
        data.metrics.saccadeCountMeans, ...
        data.metrics.saccadeAverageAmplitudeMeans, ...
        data.metrics.fixationAverageDurationMeans] = ...
    PrepareBoxChartDataCondition(data.metrics.responseTime, ...
        data.metrics.saccades, ...
        data.metrics.fixations, ...
        trialTable, ...
        conditions, ...
        constants);

ehmiGroupC = reshape(repmat(reshape(repmat( ...
    categorical(conditions.EHMI),1,numel(conditions.MASK))',1,[]), ...
        constants.NUMPARTICIPANTS,1),[],1);
maskGroupC = reshape(repmat(repmat( ...
    categorical(conditions.MASK)',1,numel(conditions.EHMI)), ...
    constants.NUMPARTICIPANTS,1),[],1);
```

Listing 5: `Main_PrepareData.m`

```matlab
%% Main_Plot

close all hidden

% Plot example of raw vs prepared data.
participant = 16;
trial = 48;
[figureAxes,figureNames] = PlotRawVsPrepared( ...
    sampleSpan, ...
    timeSpan, ...
    data.raw.time, ...
    data.raw.gazeX, ...
    data.raw.gazeY, ...
    data.raw.pupilSize, ...
    data.processed.gazeX, ...
    data.processed.gazeY, ...
    data.processed.pupilSize, ...
    trialOrder, ...
    participant, ...
    trial, ...
    screenSize, ...
    figureSize, ...
    constants);
for currentFigure = 1:numel(figureAxes)
    exportgraphics(figureAxes(currentFigure), ...
        fullfile(figureDir,figureNames(currentFigure)), ...
        "ContentType","vector")
end

% Plot heat maps.
[figureAxes,figureNames] = PlotHeatMaps( ...
    heatMapsSeparate, ...
    heatMapTotalRaw, ...
    heatMapTotalProcessed, ...
    screenSize, ...
    figureSize, ...
    constants);
for currentFigure = 1:numel(figureAxes)
    exportgraphics(figureAxes(currentFigure), ...
        fullfile(figureDir,figureNames(currentFigure)), ...
        "ContentType","vector")
end

% Plot response over time.
[figureAxes,figureNames] = PlotResponseTimes( ...
    timeSpan, ...
    data.metrics.keyPressesCumulativeCondition, ...
    screenSize, ...
    figureSize, ...
    conditions, ...
    constants);
for currentFigure = 1:numel(figureAxes)
    exportgraphics(figureAxes(currentFigure), ...
        fullfile(figureDir,figureNames(currentFigure)), ...
        "ContentType","vector")
end

% Plot saccades over time.
[figureAxes,figureNames] = PlotSaccades( ...
    timeSpan, ...
    data.metrics.saccadesCumulativeCondition, ...
    screenSize, ...
    figureSize, ...
    conditions, ...
    constants);
for currentFigure = 1:numel(figureAxes)
    exportgraphics(figureAxes(currentFigure), ...
        fullfile(figureDir,figureNames(currentFigure)), ...
        "ContentType","vector")
end

% Plot ellipse size over time.
[figureAxes,figureNames] = PlotEllipseSize( ...
    sampleSpan, ...
    timeSpan, ...
    data.metrics.ellipseMeanArea, ...
    data.metrics.ellipseMaxArea, ...
    screenSize, ...
    figureSize, ...
    conditions, ...
    constants);
for currentFigure = 1:numel(figureAxes)
    exportgraphics(figureAxes(currentFigure), ...
        fullfile(figureDir,figureNames(currentFigure)), ...
        "ContentType","vector")
end

% Plot eccentricity over time.
[figureAxes,figureNames] = PlotEccentricity( ...
    sampleSpan, ...
    timeSpan, ...
    data.metrics.eccentricityMean, ...
    screenSize, ...
    figureSize, ...
    conditions, ...
    constants);
for currentFigure = 1:numel(figureAxes)
    exportgraphics(figureAxes(currentFigure), ...
        fullfile(figureDir,figureNames(currentFigure)), ...
        "ContentType","vector")
```

```
end

% Plot Boxcharts of response times, saccade counts, saccade amplitudes and
% average fixation duration.
[figureAxes,figureNames] = PlotBoxCharts( ...
    ehmiGroup, ...
    maskGroup, ...
    data.metrics.ehmiResponseTimes, ...
    data.metrics.ehmiResponseTimeMeans, ...
    data.metrics.maskResponseTimes, ...
    data.metrics.maskResponseTimeMeans, ...
    data.metrics.ehmiSaccadeCounts, ...
    data.metrics.ehmiSaccadeCountsMeans, ...
    data.metrics.maskSaccadeCounts, ...
    data.metrics.maskSaccadeCountsMeans, ...
    data.metrics.ehmiSaccadeAverageAmplitudes, ...
    data.metrics.ehmiSaccadeAverageAmplitudesMeans, ...
    data.metrics.maskSaccadeAverageAmplitudes, ...
    data.metrics.maskSaccadeAverageAmplitudesMeans, ...
    data.metrics.ehmiFixationAverageDurations, ...
    data.metrics.ehmiFixationAverageDurationsMeans, ...
    data.metrics.maskFixationAverageDurations, ...
    data.metrics.maskFixationAverageDurationsMeans, ...
    screenSize, ...
    figureSize);
for currentFigure = 1:numel(figureAxes)
    exportgraphics(figureAxes(currentFigure), ...
        fullfile(figureDir,figureNames(currentFigure)), ...
        "ContentType","vector")
end

% Plot scatter plots of response times, saccade counts, saccade amplitudes
% and average fixation duration.
[figureAxes,figureNames] = PlotScatterPlots( ...
    ehmiGroup, ...
    maskGroup, ...
    data.metrics.ehmiResponseTimes, ...
    data.metrics.maskResponseTimes, ...
    data.metrics.ehmiSaccadeCounts, ...
    data.metrics.maskSaccadeCounts, ...
    data.metrics.ehmiSaccadeAverageAmplitudes, ...
    data.metrics.maskSaccadeAverageAmplitudes, ...
    data.metrics.ehmiFixationAverageDurations, ...
    data.metrics.maskFixationAverageDurations, ...
    screenSize, ...
    figureSize, ...
    conditions);
for currentFigure = 1:numel(figureAxes)
    exportgraphics(figureAxes(currentFigure), ...
        fullfile(figureDir,figureNames(currentFigure)), ...
        "ContentType","vector")
end

% Plot for between-group effect box charts
[figureAxes,figureNames] = PlotBoxChartsC( ...
        ehmiGroupC, ...
        maskGroupC, ...
        data.metrics.responseTimeMeans, ...
        data.metrics.saccadeCountMeans, ...
        data.metrics.saccadeAverageAmplitudeMeans, ...
        data.metrics.fixationAverageDurationMeans, ...
        screenSize, ...
        figureSize, ...
        conditions);
for currentFigure = 1:numel(figureAxes)
    exportgraphics(figureAxes(currentFigure), ...
        fullfile(figureDir,figureNames(currentFigure)), ...
        "ContentType","vector")
end
```

Listing 6: `Main_Plot.m`

```
%% Main_Redraw
% Interpolate to fit the data to the stimulus frame count.
[data.interpolated.gazeX, ...
        data.interpolated.gazeY, ...
        data.interpolated.pupilSize, ...
        data.interpolated.meanX, ...
        data.interpolated.meanY, ...
        data.interpolated.stdXPrime, ...
        data.interpolated.stdYPrime, ...
        data.interpolated.theta] = ...
    InterpolateData( ...
        data.processed.gazeX, ...
        data.processed.gazeY, ...
        data.processed.pupilSize, ...
        data.metrics.meanX, ...
        data.metrics.meanY, ...
        data.metrics.stdXPrime, ...
        data.metrics.stdYPrime, ...
        data.metrics.theta, ...
        sampleSpan, ...
        timeSpan, ...
        constants);

% Sample key presses to use as markers.
data.interpolated.keyPressesSampled = ...
    SampleKeyPresses( ...
        timeSpan, ...
        data.metrics.keyPresses, ...
        constants);

% Draw gaze on stimuli.
% Every trial takes between ~25-30 seconds to process.
drawEllipse = false;
stimulusGreyScale = false;
for trial = 1:constants.NUMTRIALS
    DrawGazeOnStimulus( ...
        trial, ...
        data.interpolated.gazeX, ...
        data.interpolated.gazeY, ...
        data.interpolated.meanX, ...
        data.interpolated.meanY, ...
        data.interpolated.stdXPrime, ...
        data.interpolated.stdYPrime, ...
        data.interpolated.theta, ...
        scaleFactor, ...
        trialTable, ...
        data.interpolated.keyPressesSampled, ...
        drawEllipse, ...
        stimulusGreyScale, ...
        constants)
end

% Recreate trials.
% Redraw the blur around the gaze mask, reconstructing the trial as it was
% experienced by the participant.
participant = 1;
for trial = [1,23,45]
    ReconstructTrial( ...
        trial, ...
        participant, ...
        data.interpolated.gazeX, ...
        data.interpolated.gazeY, ...
        trialTable, ...
        constants)
end
```

Listing 7: `Main_Redraw.m`

C.1.2. FUNCTIONS

```
function [time, ...
        gazeX, ...
        gazeY, ...
        pupilSize, ...
        trialOrder] = ...
    ImportFromEdfs( ...
        edfList, ...
        trialTable, ...
        constants)

% This function processess all *.edf files present in edfList.
%
% Inputs
% - edfList is a struct containing the fullfile information to all .edf
%   files.
% - trialTable is a nx3 table, containing the initial order of the trials,
%   where n is the number of trials. The columns contain the information to
%   identify the randomized trial number used in the EyeLink
%   ExperimentBuilder to this fixed order.
% - constants is a struct containing various global constants.
%
% Outputs
% - time, gazeX, gazeY, pupilSize are nxm cell matrices, where n equals
%   the number of trials and m equals the number of participants. Each cell
%   contains nxm Arrays containing the raw sampled data for each of the
%   individual trials per participant, where n and m can vary.
% - trialOrder is an nxm matrix where n equals the number of trials and m
%   equals the number of participants. It contains the fixed trial number
%   as represented by trialTable.

[time,gazeX,gazeY,pupilSize] = ...
    deal(cell(constants.NUMTRIALS,constants.NUMPARTICIPANTS));

trialOrder = zeros(constants.NUMTRIALS,constants.NUMPARTICIPANTS);

for participant = 1:constants.NUMPARTICIPANTS
    disp(strcat("Extracting .edf ",string(participant),"/", ...
        string(constants.NUMPARTICIPANTS),"."))

    tic
    % Import edf into struct
    edfStruct = Edf2Mat(fullfile(edfList(participant).folder, ...
        edfList(participant).name));

    % Extract and parse each trial
    trialIndex = [find(ismember(edfStruct.Events.Messages.time, ...
            edfStruct.Events.Start.time))';
        length(edfStruct.Events.Messages.time)];
    trialIndex = trialIndex(1+constants.NUMTESTTRIALS:end);

    trialFrameCount = zeros(length(trialIndex)-1,1);
    for trial = 1:constants.NUMTRIALS
        searchRange = trialIndex(trial):trialIndex(trial+1)-1;
        startFrameTime = edfStruct.Events.Messages.time( ...
            searchRange(find(endsWith(edfStruct.Events.Messages.info( ...
                searchRange),"DISPLAY_START"),1)));
        endFrameTime = edfStruct.Events.Messages.time( ...
            searchRange(find(endsWith(edfStruct.Events.Messages.info( ...
                searchRange),"DISPLAY_END"),1)));
        startFrameIndex = find( ...
            edfStruct.Samples.time==startFrameTime,1,'last');
        endFrameIndex = find( ...
            edfStruct.Samples.time==endFrameTime,1,'last');
        trialFrameCount(trial,:) = endFrameTime-startFrameTime;

        trialIdent = cell2table([str2double( ...
            extractAfter(edfStruct.Events.Messages.info(searchRange( ...
                contains(edfStruct.Events.Messages.info(searchRange), ...
                    "TRIAL_VAR preset ")))',"preset ")), ...
            extractAfter(edfStruct.Events.Messages.info(searchRange( ...
                contains(edfStruct.Events.Messages.info(searchRange), ...
                    "TRIAL_VAR ehmi_type ")))',"ehmi_type "), ...
            extractAfter(edfStruct.Events.Messages.info(searchRange( ...
                contains(edfStruct.Events.Messages.info(searchRange), ...
                    "TRIAL_VAR mask ")))',"mask ")], ...
            'VariableNames',trialTable.Properties.VariableNames);

        [~,trialOrder(trial,participant)] = ...
            ismember(trialIdent,trialTable);

        time{trial,participant} = ...
            edfStruct.Samples.time(startFrameIndex:endFrameIndex,:);
        gazeX{trial,participant} = ...
            edfStruct.Samples.gx(startFrameIndex:endFrameIndex,:);
        gazeY{trial,participant} = ...
            edfStruct.Samples.gy(startFrameIndex:endFrameIndex,:);
        pupilSize{trial,participant} = ...
            edfStruct.Samples.pa(startFrameIndex:endFrameIndex,:);
    end
    clear edfStruct
    toc
end

end
```

Listing 8: ImportFromEdfs.m

```
function keyPressTables = ...
    ImportFromTxts( ...
```

```matlab
        txtList, ...
        trialTable, ...
        constants)

% This function processess all RESULTS_FILE.txt files present in txtList.
%
% Inputs
% - txtList is a struct containing the fullfile information to all
%   RESULTS_FILE.txt files.
% - trialTable is a nxm table, containing the desired order of the trials,
%   where n is the number of trials, and m is the number of identifier
%   categories.
% - constants is a struct containing various global constants.
%
% Outputs
% - keyPressTables is a 1xm cell array, where m is the number of
%   participants. Every column represents one participant, and every cell
%   in this column represents the keyPressTable associated with this
%   participant.
%
% RESULTS_FILE.txt is a file generated by the EyeLink Experiment. It
% contains event data in the following named columns:
%   [Session_Name_,Trial_Index_,Trial_Recycled_,preset,ehmi_type,mask, ...
%   KEY_PRESS_TIME,DISPLAY_START_TIME,RESPONSE_TIME,KEY_TRIGGER,type]
%
% The 4th, 5th, 6th, 7th and 8th columns are taken:
%   [preset,ehmi_type,mask,KEY_PRESS_TIME,DISPLAY_START_TIME],
% and the variable names are renamed to:
%   [PRESET,EHMI,MASK,TIMEKEYPRESS,TIMEDISPLAYSTART]
% TIMEKEYPRESS and TIMEDISPLAYSTART are global time values in [ms].

tic

keyPressTables = deal(cell(1,constants.NUMPARTICIPANTS));

for participant = 1:constants.NUMPARTICIPANTS
    keyPressTable = readtable(fullfile(txtList(participant).folder, ...
        txtList(participant).name));
    keyPressTable = keyPressTable(:,4:8);
    keyPressTable.Properties.VariableNames = ...
        [trialTable.Properties.VariableNames, ...
            'TIMEKEYPRESS','TIMEDISPLAYSTART'];
    keyPressTables{participant} = keyPressTable;
end

toc

end
```

Listing 9: `ImportFromTxts.m`

```matlab
function [timeNew, ...
        gazeXNew, ...
        gazeYNew, ...
        pupilSizeNew, ...
        fixations, ...
        saccades] = ...
    PrepareEdfData( ...
        timeOld, ...
        gazeXOld, ...
        gazeYOld, ...
        pupilSizeOld, ...
        trialOrder, ...
        constants)

% This function touches the raw .edf data. First it resorts the data
% according to trialOrder. Second it feeds the sorted data to ProcessEyes,
% resulting in smoothed fixed data points. Third, the array lengths are
% equalized to the number of samples in the trial duration.
%
% Inputs
% - timeOld, gazeXOld, gazeYOld, pupilSizeOld are nxm cell matrices(*)
%   containing their respective data.
% - trialOrder is an nxm matrix(*) containing the actual trial indices for
%   each randomized trial.
% - constants is a struct containing various global constants.
%
% Outputs
% - timeNew, gazeXNew, gazeYNew, pupilSizeNew are nxm matrices(*)
%   containing their respective data, sorted by trialOrder.
% - fixations, saccades are nxm matrices(*) containing fixation and
%   saccade data output by ProcessEye, sorted by trialOrder.
%
% * n equals the number of trials and m equals the number of participants.

disp("Resorting and touching extracted .edf data.")

tic

[timeNew,gazeXNew,gazeYNew,pupilSizeNew,fixations,saccades] = ...
    deal(cell(constants.NUMTRIALS,constants.NUMPARTICIPANTS));

% Get time and averaged gaze and pupil size data from raw sampled data, and
% resort them to all participants' trials by trial number.
for participant = 1:constants.NUMPARTICIPANTS
    for trial = 1:constants.NUMTRIALS
        timeNew(trialOrder(trial,participant),participant) = ...
            timeOld(trial,participant);
        gazeXNew{trialOrder(trial,participant),participant} = ...
            (gazeXOld{trial,participant}(:,1)+ ...
            gazeXOld{trial,participant}(:,2))/2;
        gazeYNew{trialOrder(trial,participant),participant} = ...
            (gazeYOld{trial,participant}(:,1)+ ...
            gazeYOld{trial,participant}(:,2))/2;
        pupilSizeNew{trialOrder(trial,participant),participant} = ...
            (pupilSizeOld{trial,participant}(:,1)+ ...
            pupilSizeOld{trial,participant}(:,2))/2;
    end
end

% Touch data. This function takes the sorted data, removes blinks and
% half-blinks, then smooths it with a median filter. Saccades and fixations
% are also identified here, and output into their respective variables.
% Then, the length of the arrays is padded and cut to match the stimulus
% duration.
for participant = 1:constants.NUMPARTICIPANTS
    for trial = 1:constants.NUMTRIALS
        [fixations{trial,participant},saccades{trial,participant}, ...
                gazeXNew{trial,participant}, ...
                gazeYNew{trial,participant}, ...
                pupilSizeNew{trial,participant},~] = ...
            ProcessEyes( ...
                gazeXNew{trial,participant}, ...
                gazeYNew{trial,participant}, ...
                pupilSizeNew{trial,participant});

        gazeXNew{trial,participant}( ...
            end+1:constants.SAMPLERATE*( ...
                constants.DURATION+constants.PADDEDDURATION)) = nan;
        gazeYNew{trial,participant}( ...
            end+1:constants.SAMPLERATE*( ...
                constants.DURATION+constants.PADDEDDURATION)) = nan;
        pupilSizeNew{trial,participant}( ...
            end+1:constants.SAMPLERATE*( ...
                constants.DURATION+constants.PADDEDDURATION)) = nan;
        gazeXNew{trial,participant} = ...
            gazeXNew{trial,participant}( ...
                1+constants.SAMPLERATE*constants.PADDEDDURATION: ...
                constants.SAMPLERATE*( ...
                    constants.DURATION+constants.PADDEDDURATION));
        gazeYNew{trial,participant} = ...
            gazeYNew{trial,participant}( ...
                1+constants.SAMPLERATE*constants.PADDEDDURATION: ...
                constants.SAMPLERATE*( ...
                    constants.DURATION+constants.PADDEDDURATION));
        pupilSizeNew{trial,participant} = ...
            pupilSizeNew{trial,participant}( ...
                1+constants.SAMPLERATE*constants.PADDEDDURATION: ...
                constants.SAMPLERATE*( ...
                    constants.DURATION+constants.PADDEDDURATION));
    end
end

toc

end

```

Listing 10: PrepareEdfData.m

```
function keyPresses = PrepareTxtData(keyPressTables,trialTable,constants)

% This function takes all the keyPressTables and matches them against the
% trials represented by trialTable.
%
% Inputs
% - keyPressTables is a 1xm cell array, where m is the number of
%   participants. Every column represents one participant, and every cell
%   in this column represents the key press events table associated with
%   this participant. Every key press events table contains the following
%   columns: [PRESET,EHMI,MASK,TIMEKEYPRESS,TIMEDISPLAYSTART], where
%   PRESET, EHMI, MASK are trial identifiers, and TIMEKEYPRESS and
%   TIMEDISPLAYSTART are global time values in [ms].
% - trialTable is a nxm table, containing the desired order of the trials,
%   where n is the number of trials, and m is the number of identifier
%   categories.
% - constants is a struct containing various global constants.
%
% Outputs
% - keyPresses is an nxm cell matrix, where n equals the number of trials,
%   and m equals the number of participants. Every cell contains
%   keyPressSequence, which is an nx2 matrix, where n equals the number of
%   key press events. Every row contains [time,event]. For every nth odd
%   event we assume event = constants.KEYPRESSED = 1, and every nth even
%   event we assume event = constants.KEYRELEASED = -1.

disp("Resorting and touching extracted RESULTS_FILE.txt data.")

tic

keyPresses = deal(cell(constants.NUMTRIALS,constants.NUMPARTICIPANTS));
timeShift = constants.SIGNALFRAME/constants.FRAMERATE;

for participant = 1:constants.NUMPARTICIPANTS
    [~,resultsOrder] = ismember(keyPressTables{participant}( ...
        :,trialTable.Properties.VariableNames),trialTable);
    for trial = 1:constants.NUMTRIALS
        matches = find(resultsOrder==trial);
        keyPressSequence = zeros(length(matches)+1,2);
        keyPressSequence(1,:) = [-constants.PADDEDDURATION-timeShift,0];
        for match = 1:length(matches)
            if mod(match,2) % odd
                keyPressSequence(match+1,:) = ...
                    [(keyPressTables{participant}.TIMEKEYPRESS( ...
                            matches(match))- ...
                        keyPressTables{participant}.TIMEDISPLAYSTART( ...
                            matches(match)))/ ...
                        1000-constants.PADDEDDURATION-timeShift, ...
                    constants.KEYPRESSED];
            else % even
                keyPressSequence(match+1,:) = ...
                    [(keyPressTables{participant}.TIMEKEYPRESS( ...
                            matches(match))- ...
                        keyPressTables{participant}.TIMEDISPLAYSTART( ...
                            matches(match)))/ ...
                        1000-constants.PADDEDDURATION-timeShift, ...
                    constants.KEYRELEASED];
            end
        end
        keyPresses{trial,participant} = keyPressSequence;
    end
end

toc

end
```

Listing 11: PrepareTxtData.m

```
function keyPressesCumulative = AccumulateKeyPresses(keyPresses, constants)

% This function accumulates all key presses present in the keyPresses cell
% per trial.
%
% Inputs
% - keyPresses is an nxm cell matrix, where n equals the number of trials,
%   and m equals the number of participants. Each cell contains an array
%   with the key press data for that participant's trial.
% - constants is a struct containing various global constants.
%
% Outputs
% - keyPressesCumulative is an nx1 cell array, where n equals the number of
%   trials. Each cell contains the cumulative key press data for that
%   trial.

keyPressesCumulative = deal(cell(constants.NUMTRIALS,1));
timeShift = constants.SIGNALFRAME/constants.FRAMERATE;

% Cumulative sums per trial
for trial = 1:constants.NUMTRIALS
    keyPressesCumulative{trial} = ...
        sortrows(cat(1,keyPresses{trial,:}))';
    keyPressesCumulative{trial} = ...
        keyPressesCumulative{trial}(:,constants.NUMPARTICIPANTS:end);
    keyPressesCumulative{trial}(2,:) = ...
        cumsum(keyPressesCumulative{trial}(2,:));
    keyPressesCumulative{trial}(:,end) = ...
        [constants.DURATION-timeShift;
        keyPressesCumulative{trial}(2,end-1)];
    keyPressesCumulative{trial}(1,:) = ...
        keyPressesCumulative{trial}(1,:);
end

end
```

Listing 12: AccumulateKeyPresses.m

```
function keyPressesCumulativeCondition = ...
    AccumulateKeyPressesConditions( ...
        keyPresses, ...
        trialTable, ...
        conditions, ...
        constants)

% This function accumulates all key presses present in the keyPresses cell
% grouped per condition.
%
% Inputs
% - keyPresses is an nxm cell matrix, where n equals the number of trials,
%   and m equals the number of participants. Each cell contains an array
%   with the key press data for that participant's trial.
% - trialTable, nx3 table, containing the preset number and independent
%   variables for all n trials.
% - conditions, 1x1 struct containing i jx1 categorical arrays, definining
%   the j conditions for each of the i independent variables.
% - constants is a struct containing various global constants.
%
% Outputs
% - keyPressesCumulativeCondition is a (i*j)x1 cell array, where i*j equals
%   the total number of conditions. Each cell contains the cumulative key
%   press data for that condition.

keyPressesCumulativeCondition = ...
    deal(cell(numel(conditions.EHMI)+numel(conditions.MASK),1));
timeShift = constants.SIGNALFRAME/constants.FRAMERATE;

% Cumulative sums per condition
for condition = 1:numel(conditions.EHMI)+numel(conditions.MASK)
    if ~(condition>numel(conditions.EHMI))
        trials = find(trialTable.PRESET~=7& ...
            trialTable.EHMI==conditions.EHMI(condition));
    else
        trials = find(trialTable.PRESET~=7& ...
            trialTable.MASK==conditions.MASK( ...
                condition-numel(conditions.EHMI)));
    end

    for trial = 1:numel(trials)
        keyPressesCumulativeCondition{condition} = ...
            [keyPressesCumulativeCondition{condition};
            cat(1,keyPresses{trials(trial),:})];
    end

    keyPressesCumulativeCondition{condition} = ...
        sortrows(keyPressesCumulativeCondition{condition})';
    keyPressesCumulativeCondition{condition} = ...
        keyPressesCumulativeCondition{condition}( ...
            :,constants.NUMPARTICIPANTS*numel(trials):end);
    keyPressesCumulativeCondition{condition}(2,:) = ...
        cumsum(keyPressesCumulativeCondition{condition}(2,:));
    keyPressesCumulativeCondition{condition}(:,end) = ...
        [constants.DURATION-timeShift;
        keyPressesCumulativeCondition{condition}(2,end-1)];
end

end
```

Listing 13: `AccumulateKeyPressesConditions.m`

```
function responseTime = ...
    GetResponseTime( ...
        keyPresses, ...
        constants)

% This function determines the response time for each participant for each
% trial, based on some set conditions. Only responses made after signal
% frame, with a certain minimum response time are considered valid. If no
% valid response time can be determined, the response is NaN.
%
% Inputs
% - keyPresses is an nxm cell matrix, where n equals the number of trials,
%   and m equals the number of participants. Each cell contains an array
%   with key press data for that participant's trial.
% - constants is a struct containing various global constants.
%
% Outputs
% - responseTime is an nxm matrix, where n equals the number of trials, and
%   m equals the number of participants. The values in the matrix represent
%   the response time for that participant's trial.

responseTime = deal(nan(constants.NUMTRIALS,constants.NUMPARTICIPANTS));

for participant = 1:constants.NUMPARTICIPANTS
    for trial = 1:constants.NUMTRIALS
        response = keyPresses{trial,participant}(find( ...
            keyPresses{trial,participant}(:,1)> ...
                constants.MINREACTIONTIME& ...
            keyPresses{trial,participant}(:,2)==1,1));
        if ~isempty(response)
            responseTime(trial,participant) = response;
        end
    end
end

end
```

Listing 14: GetResponseTime.m

```
function heatMap = PrepareHeatMap(gazeX,gazeY,gridSize,constants)

% This function creates a 2D histogram or heatmap of the provided gaze
% data. gridSize can be used to create larger bins.
%
% Inputs
% - gazeX, gazeY are nxm cell matrices, where n equals the number of trials
%   and m equals the number of participants. Each cell contains a jx1
%   array, where j equals the number of data points sampled at the
%   framerate of the stimuli. They contain the gaze data for each trial of
%   each of the participants.
% - gridSize is an int, used to scale the bin sizes.
% - constants is a struct containing various global constants.
%
% Outputs
% - heatMap, an nxm matrix, where n is the vertical number of bins, and m
%   is the horizontal number of bins.

binsX = linspace(1,constants.SCREENWIDTH, ...
    constants.SCREENWIDTH/gridSize+1);
binsY = linspace(1,constants.SCREENHEIGHT, ...
    constants.SCREENHEIGHT/gridSize+1);
heatMap = histcounts2(floor(gazeY),floor(gazeX),binsY,binsX);

end
```

Listing 15: PrepareHeatMap.m

```
function [meanX, ...
        meanY, ...
        stdXPrime, ...
        stdYPrime, ...
        theta] = ...
    CalculateDispersion( ...
        gazeX, ...
        gazeY, ...
        constants)

% This function calculates various metrics, used to describe the dispersion
% of the point cloud.
%
% References
% - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4358977/
% - https://www.jstor.org/stable/490885
%
% Inputs
% - gazeX, gazeY are nxm cell matrices containing the gaze data, where n
%   equals the number of trials, and m equals the number of participants.
% - constants is a struct containing various global constants.
%
% Outputs
% - meanX, meanY, stdXPrime, stdYPrime, theta are nxi matrices, where n
%   equals the number of trials and i equals the number of samples.

disp("Calculating dispersion for all trials' samples.")

tic

[meanX,meanY,stdXPrime,stdYPrime,theta] = ...
    deal(nan(constants.NUMTRIALS,constants.SAMPLERATE*constants.DURATION));

% Calculate means and dispersion for each sample of each trial.
for trial = 1:constants.NUMTRIALS
    gazeXTemp = [gazeX{trial,:}]';
    gazeYTemp = [gazeY{trial,:}]';

    meanX(trial,:) = mean(gazeXTemp,1,"omitnan");
    meanY(trial,:) = mean(gazeYTemp,1,"omitnan");

    xDash = gazeXTemp-meanX(trial,:);
    yDash = gazeYTemp-meanY(trial,:);

    A = sum(xDash.^2,1)-sum(yDash.^2,1);
    B = sqrt(A.^2+4*sum(xDash.*yDash,1).^2);
    C = 2*sum(xDash.*yDash,1);

    % Standard ROT(th) = [cos(th), -sin(th); sin(th), cos(th)]
    % Rotates points in the xy-Cartesian plane (x positive to the right, y
    % positive up) counterclockwise through an angle th about the origin of
    % the Cartesian coordinate system.
    %
    % We have a non-standard plane, with x positive to the right, but y
    % positive down. The standard ROT will rotate points clockwise through
    % an angle th about the origin of our screen coordinate system. We will
    % assume clockwise rotations so standard ROT matrices can be used.
    theta(trial,:) = atan((A+B)./C);
    stdXPrime(trial,:) = sqrt(mean((xDash.*cos(theta(trial,:))- ...
        yDash.*sin(theta(trial,:))).^2,1));
    stdYPrime(trial,:) = sqrt(mean((xDash.*sin(theta(trial,:))+ ...
        yDash.*cos(theta(trial,:))).^2,1));
end

toc

end
```

Listing 16: `CalculateDispersion.m`

```
function e = CalculateEccentricity(ax1,ax2)

% This function calculates the eccentricity of an ellipse according to
% eccentricty e = sqrt(1-axMin^2/axMaj^2).
%
% Inputs
% - ax1, the first of the primary axes of the point cloud. Can be either a
%   double or an array of doubles.
% - ax2, the second of the primary axes of the point cloud. Can be either a
%   double or an array of doubles.
%
% Outputs
% - e, the eccentricity of the ellipse. If inputs are double, output is
%   double. If inputs are an array of doubles, outputs an array of doubles.

% Determine ellipse's major and minor axes.
axMaj = max(ax1,ax2);
axMin = min(ax1,ax2);

% Eccentricity e
e = sqrt(1-(axMin.^2./axMaj.^2));

end
```

Listing 17: `CalculateEccentricity.m`

```
function A = CalculateEllipseArea(ax1,ax2)

% This function calculates the area of an ellipse according to area
% A = pi*a*b, where a and b are the length of the two primary axes.
%
% Inputs
% - ax1, the first of the primary axes of the point cloud. Can be either a
%   double or an array of doubles.
% - ax2, the second of the primary axes of the point cloud. Can be either a
%   double or an array of doubles.
%
% Outputs
% - A, the area of the ellipse. If inputs are double, output is
%   double. If inputs are an array of doubles, outputs an array of doubles.

A = pi*ax1.*ax2;

end
```

Listing 18: `CalculateEllipseArea.m`

```
function [eccentricityMean, ...
        ellipseMeanArea] = ...
    PrepareEllipseData( ...
        eccentricity, ...
        ellipseArea, ...
        trialTable, ...
        conditions, ...
        constants)

% This function calculates two characteristics of the deviational ellipse
% for each of the six individual conditions: the mean eccentricity, and the
% mean area.
%
% Inputs
% - eccentricity, ellipseArea are nxi matrices, where n equals the number
%   of trials and i equals the number of data points sampled at the
%   frequency of the EyeLink.
% - trialTable, nx3 table, containing the preset number and independent
%   variables for all n trials.
% - conditions, 1x1 struct containing i jx1 categorical arrays, defining
%   the j conditions for each of the i independent variables.
% - constants is a struct containing various global constants.
%
% Outputs
% - eccentricityMean, ellipseMeanArea are mxi matrices, where m equals the
%   number of conditions (of both independent variables combined) and i
%   equals the number of data points sampled at the frequency of the
%   EyeLink.

[eccentricityMean,ellipseMeanArea] = ...
    deal(zeros(numel(conditions.EHMI)+numel(conditions.MASK), ...
    constants.SAMPLERATE*constants.DURATION));

for condition = 1:numel(conditions.EHMI)+numel(conditions.MASK)
    if condition <= numel(conditions.EHMI)
        ellipseMeanArea(condition,:) = mean(ellipseArea( ...
            trialTable.EHMI==conditions.EHMI(condition),:));
        eccentricityMean(condition,:) = mean(eccentricity( ...
            trialTable.EHMI==conditions.EHMI(condition),:));
    else
        ellipseMeanArea(condition,:) = mean(ellipseArea( ...
            trialTable.MASK==conditions.MASK(1+mod( ...
                condition-1,numel(conditions.EHMI))),:));
        eccentricityMean(condition,:) = mean(eccentricity( ...
            trialTable.MASK==conditions.MASK(1+mod( ...
                condition-1,numel(conditions.EHMI))),:));
    end
end

end
```

Listing 19: PrepareEllipseData.m

```
function [ehmiResponseTimes, ...
        maskResponseTimes, ...
        ehmiSaccadeCounts, ...
        maskSaccadeCounts, ...
        ehmiSaccadeAverageAmplitudes, ...
        maskSaccadeAverageAmplitudes, ...
        ehmiFixationAverageDurations, ...
        maskFixationAverageDurations] = ...
    PrepareBoxChartData( ...
        responseTime, ...
        saccades, ...
        fixations, ...
        trialTable, ...
        conditions, ...
        constants)

% This function restructures the data so that it can be used for
% statistical analysis and to create box charts and scatter plots. The
% resulting arrays can be used to analyse the within-group effects.
%
% Inputs
% - responseTime is an nxm matrix, where n equals the number of trials, and
%   m equals the number of participants. The values in the matrix represent
%   the response time for that participant's trial.
% - saccades, fixations are nxm cell matrices, where n equals the number of
%   trials, and m equals the number of participants. Each cell contains
%   saccade and fixation data respectively for each trial of each
%   participant.
% - trialTable, nx3 table, containing the preset number and independent
%   variables for all n trials.
% - conditions, 1x1 struct containing i jx1 categorical arrays, definining
%   the j conditions for each of the i independent variables.
% - constants is a struct containing various global constants.
```

```
%
% Outputs
% - ehmiResponseTimes, maskResponseTimes, ehmiSaccadeCounts,
%   maskSaccadeCounts, ehmiSaccadeAverageAmplitudes,
%   maskSaccadeAverageAmplitudes, ehmiFixationAverageDurations,
%   maskFixationAverageDurations are px1 arrays where p equals the number
%   of participants times the conditions of a single independent variable.

% Fixation frequency and duration, and saccade amplitude, per condition.
% Boxplots are constructed from averaged values per condition: (1 value
% per participant).
[saccadeCount,saccadeAverageAmplitude,fixationAverageDuration] = ...
    deal(zeros(constants.NUMTRIALS,constants.NUMPARTICIPANTS));

% Extract values from saccade and fixation matrices.
for trial = 1:constants.NUMTRIALS
    for participant = 1:constants.NUMPARTICIPANTS
        saccadeCount(trial,participant) = ...
            size(saccades{trial,participant},1);

        for saccade = 1:saccadeCount(trial,participant)
            saccadeAverageAmplitude(trial,participant) = ...
                saccadeAverageAmplitude(trial,participant)+ ...
                saccades{trial,participant}(saccade,3);
            fixationAverageDuration(trial,participant) = ...
                fixationAverageDuration(trial,participant)+ ...
                fixations{trial,participant}(saccade,3);
        end
        saccadeAverageAmplitude(trial,participant) = ...
            saccadeAverageAmplitude(trial,participant)/ ...
            saccadeCount(trial,participant);
        fixationAverageDuration(trial,participant) = ...
            fixationAverageDuration(trial,participant)/ ...
            saccadeCount(trial,participant);
    end
end

[responseTimeMeans,saccadeCountMeans, ...
        saccadeAverageAmplitudeMeans,fixationAverageDurationMeans] = ...
    deal(zeros(numel(conditions.EHMI)+numel(conditions.MASK), ...
        constants.NUMPARTICIPANTS));

% Calculate average per condition for each participant.
for participant = 1:constants.NUMPARTICIPANTS
    for condition = 1:numel(conditions.EHMI)+numel(conditions.MASK)
        if condition <= numel(conditions.EHMI)
            responseTimeMeans(condition,participant) = ...
                mean(responseTime(~(trialTable.PRESET==7)& ...
                    trialTable.EHMI==conditions.EHMI(condition), ...
                        participant),"omitnan");
            saccadeCountMeans(condition,participant) = ...
                mean(saccadeCount(~(trialTable.PRESET==7)& ...
                    trialTable.EHMI==conditions.EHMI(condition), ...
                        participant),"omitnan");
            saccadeAverageAmplitudeMeans(condition,participant) = ...
                mean(saccadeAverageAmplitude(~(trialTable.PRESET==7)& ...
                    trialTable.EHMI==conditions.EHMI(condition), ...
                        participant),"omitnan");
            fixationAverageDurationMeans(condition,participant) = ...
                mean(fixationAverageDuration(~(trialTable.PRESET==7)& ...
                    trialTable.EHMI==conditions.EHMI(condition), ...
```

```
                        participant),"omitnan");
        else
            responseTimeMeans(condition,participant) = ...
                mean(responseTime(~(trialTable.PRESET==7)& ...
                    trialTable.MASK==conditions.MASK(1+mod(condition-1, ...
                        numel(conditions.MASK))),participant),"omitnan");
            saccadeCountMeans(condition,participant) = ...
                mean(saccadeCount(~(trialTable.PRESET==7)& ...
                    trialTable.MASK==conditions.MASK(1+mod(condition-1, ...
                        numel(conditions.MASK))),participant),"omitnan");
            saccadeAverageAmplitudeMeans(condition,participant) = ...
                mean(saccadeAverageAmplitude(~(trialTable.PRESET==7)& ...
                    trialTable.MASK==conditions.MASK(1+mod(condition-1, ...
                        numel(conditions.MASK))),participant),"omitnan");
            fixationAverageDurationMeans(condition,participant) = ...
                mean(fixationAverageDuration(~(trialTable.PRESET==7)& ...
                    trialTable.MASK==conditions.MASK(1+mod(condition-1, ...
                        numel(conditions.MASK))),participant),"omitnan");
        end
    end
end

% Reshape vectors.
ehmiResponseTimes = reshape( ...
    responseTimeMeans(1:numel(conditions.EHMI),:),[],1);
maskResponseTimes = reshape( ...
    responseTimeMeans(numel(conditions.EHMI)+1:end,:),[],1);

ehmiSaccadeCounts = reshape( ...
    saccadeCountMeans(1:numel(conditions.EHMI),:),[],1);
maskSaccadeCounts = reshape( ...
    saccadeCountMeans(numel(conditions.EHMI)+1:end,:),[],1);

ehmiSaccadeAverageAmplitudes = reshape( ...
    saccadeAverageAmplitudeMeans(1:numel(conditions.EHMI),:),[],1);
maskSaccadeAverageAmplitudes = reshape( ...
    saccadeAverageAmplitudeMeans(numel(conditions.EHMI)+1:end,:),[],1);

ehmiFixationAverageDurations = reshape( ...
    fixationAverageDurationMeans(1:numel(conditions.EHMI),:),[],1);
maskFixationAverageDurations = reshape( ...
    fixationAverageDurationMeans(numel(conditions.EHMI)+1:end,:),[],1);

end
```

Listing 20: `PrepareBoxChartData.m`

```
function [anovaTable, ...
        multcompareTable] = ...
    StatisticalTests( ...
        metric, ...
        groups)

% This function performs an ANOVA and multcompare post-hoc test
% with Bonferroni correction on the provided metric.
%
% Inputs
% - metric, must be a nx1 matrix, where n is the number of trials. The
%   metric being analysed.
% - groups, must be a nx1 categorical array, where n is the number of
```

```matlab
%   trials. The group to which the metric is matched.
%
% Outputs
% - anovaTable, a table containing the ANOVA output.
% - multcompareTable, a table containing the multcompare post-hoc test
%   output.

% Perform an anova, followed by a pairwise comparison of its test results,
% which allows for post-hoc corrections.
if ~iscell(groups)
    [~,atab,stats] = anova1(metric,groups,"off");

    varNames = ["ss","df","ms","ff","probf"];

    [c,~,~,g] = multcompare(stats, ...
        "CriticalValueType","bonferroni", ...
        "Display","off");
else
    [~,atab,stats,~] = anovan(metric,groups, ...
        "model","interaction", ...
        "display","off");

    varNames = ["ss","df","sing","ms","ff","probf"];

    [c,~,~,g] = multcompare(stats, ...
        "Dimension",[1,2], ...
        "CriticalValueType","bonferroni", ...
        "Display","off");
end

anovaTable = cell2table(atab(2:end,2:end));
anovaTable.Properties.VariableNames = varNames;
anovaTable.Properties.RowNames = atab(2:end,1);
anovaTable.Properties.DimensionNames{1} = lower(atab{1,1});

varNames = ["cia","diff","cib","pp"];

g = regexprep(g,"X\d=","");
g = regexprep(g,",","+");

multcompareTable = array2table(c(:,3:end), ...
    "VariableNames",varNames, ...
    "RowNames",strcat(string(g(c(:,1))),"-",string(g(c(:,2)))));
multcompareTable.Properties.DimensionNames{1} = 'conditions';

end
```

Listing 21: `StatisticalTests.m`

```matlab
function [figureAxes,figureNames] = ...
    PlotRawVsPrepared( ...
        sampleSpan, ...
        timeSpan, ...
        timeRaw, ...
        gazeXRaw, ...
        gazeYRaw, ...
        pupilSizeRaw, ...
        gazeXProcessed, ...
        gazeYProcessed, ...
        pupilSizeProcessed, ...
        trialOrder, ...
        participant, ...
        trial, ...
        screenSize, ...
        figureSize, ...
        constants)

lineColors = turbo(10);

currentFigure = 1;
figureNames(currentFigure) = "rawvsproc.eps";
figureTitle = "Raw vs Processed";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/8, figureSize.*[3,2]]);
tiledlayout(2,2,"TileSpacing","compact","Padding","compact");

nexttile;
title("Raw gaze")
xlabel("Raw sample time [ms]")
yyaxis left
plottedLines = plot(timeRaw{trial,participant}, ...
    gazeXRaw{trial,participant});
set(plottedLines,{'Color'},[{lineColors(2,:)};{lineColors(3,:)}])
set(plottedLines,{'LineStyle'},[{'-'};{'-'}])
set(gca,"YColor","black")
ylim([0,constants.SCREENWIDTH])
ylabel("Screen x position [px]")
yyaxis right
plottedLines = plot(timeRaw{trial,participant}, ...
    gazeYRaw{trial,participant});
set(plottedLines,{'Color'},[{lineColors(size(lineColors,1)-2,:)}; ...
    {lineColors(size(lineColors,1)-1,:)}])
set(plottedLines,{'LineStyle'},[{'-'};{'-'}])
set(gca,"YDir","reverse")
set(gca,"YColor","black")
ylim([0,constants.SCREENHEIGHT])
ylabel("Screen y position [px]")
legend("Left x","Right x","Left y","Right y", ...
    "Location","northeastoutside")

nexttile;
title("Prepared gaze")
xlabel("t [s]")
yyaxis left
plot(sampleSpan, ...
    gazeXProcessed{trialOrder(trial,participant),participant},"blue")
xline(timeSpan(constants.SIGNALFRAME),'-',{'Signal shown'})
xline(timeSpan(constants.STOPPEDFRAME),'-',{'Car stopped'})
set(gca,"YColor","black")
xlim([sampleSpan(1),sampleSpan(end)])
ylim([0,constants.SCREENWIDTH])
ylabel("Screen x position [px]")
yyaxis right
plot(sampleSpan, ...
    gazeYProcessed{trialOrder(trial,participant),participant},"red")
set(gca,"YDir","reverse")
set(gca,"YColor","black")
xlim([sampleSpan(1),sampleSpan(end)])
ylim([0,constants.SCREENHEIGHT])
ylabel("Screen y position [px]")
legend("Combined x","Combined y","Location","northeastoutside")
```

```matlab

nexttile;
title("Raw pupil size")
xlabel("Raw sample time [ms]")
yyaxis left
plottedLines = plot(timeRaw{trial,participant}, ...
    pupilSizeRaw{trial,participant});
set(plottedLines,{'Color'},[{lineColors(2,:)};{lineColors(3,:)}]);
set(plottedLines,{'LineStyle'},[{'-'};{'-'}])
set(gca,"YColor","black")
ylim([max(max(pupilSizeRaw{trial,participant}))/1.75, ...
    max(max(pupilSizeRaw{trial,participant}))*1.1])
ylabel("Pupil area size [px^2]")
yyaxis right
set(gca,"YTick",[])
set(gca,"YColor","black")
legend("Left","Right","Location","northeastoutside")

nexttile;
title("Prepared pupil size")
xlabel("t [s]")
yyaxis left
plot(sampleSpan, ...
    pupilSizeProcessed{trialOrder(trial,participant),participant},"blue")
xline(timeSpan(constants.SIGNALFRAME),'-',{'Signal shown'})
xline(timeSpan(constants.STOPPEDFRAME),'-',{'Car stopped'})
set(gca,"YColor","black")
xlim([sampleSpan(1),sampleSpan(end)])
ylabel("Pupil diameter [mm]")
yyaxis right
set(gca,"YTick",[])
set(gca,"YColor","black")
legend("Combined","Location","northeastoutside")

end
```

Listing 22: PlotRawVsPrepared.m

```matlab
function [figureAxes,figureNames] = ...
    PlotResponseTimes( ...
        timeSpan, ...
        keyPressesCumulativeCondition, ...
        screenSize, ...
        figureSize, ...
        conditions, ...
        constants)

currentFigure = 1;
figureNames(currentFigure) = "responsetime_ehmi.eps";
figureTitle = "Number of spacebar presses per condition (a)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize.*[2,1]]);
hold on
grid on
for condition = 1:numel(conditions.EHMI)
    stairs(keyPressesCumulativeCondition{condition}(1,:), ...
        keyPressesCumulativeCondition{condition}(2,:)* ...
        constants.NUMPARTICIPANTS/100)
end
xline(timeSpan(constants.SIGNALFRAME),'-',{'Signal shown'})
xline(timeSpan(constants.STOPPEDFRAME),'-',{'Car stopped'})
legend(conditions.EHMI,"Location","southeast")
xlim([timeSpan(1)+4,timeSpan(end)])
ylim([0,100])
xlabel("Elapsed time [s]")
ylabel("Number of spacebar presses [%]")

currentFigure = currentFigure+1;
figureNames(currentFigure) = "responsetime_mask.eps";
figureTitle = "Number of spacebar presses per condition (b)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize.*[2,1]]);
hold on
grid on
for condition = 1+numel(conditions.EHMI): ...
        numel(conditions.EHMI)+numel(conditions.MASK)
    stairs(keyPressesCumulativeCondition{condition}(1,:), ...
        keyPressesCumulativeCondition{condition}(2,:)* ...
        constants.NUMPARTICIPANTS/100)
end
xline(timeSpan(constants.SIGNALFRAME),'-',{'Signal shown'})
xline(timeSpan(constants.STOPPEDFRAME),'-',{'Car stopped'})
legend(conditions.MASK,"Location","southeast")
xlim([timeSpan(1)+4,timeSpan(end)])
ylim([0,100])
xlabel("Elapsed time [s]")
ylabel("Number of spacebar presses [%]")

end
```

Listing 23: `PlotResponseTimes.m`

```
function [figureAxes,figureNames] = ...
    PlotEllipseSize( ...
        sampleSpan, ...
        timeSpan, ...
        ellipseMeanArea, ...
        ellipseMaxArea, ...
        screenSize, ...
        figureSize, ...
        conditions, ...
        constants)

currentFigure = 1;
figureNames(currentFigure) = "2sigmaarea_ehmi.eps";
figureTitle = "Mean 2*sigma clustering (described by ellipse) (a)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize.*[2,1]]);
hold on
grid on
plot(sampleSpan,ellipseMeanArea(1:3,:)/ellipseMaxArea*100)
xline(timeSpan(constants.SIGNALFRAME),'-',{'Signal shown'})
xline(timeSpan(constants.STOPPEDFRAME),'-',{'Car stopped'})
legend(conditions.EHMI,"Location","southeast")
xlim([timeSpan(1),timeSpan(end)])
ylim([0,20])
xlabel("Elapsed time [s]")
ylabel("Ellipse size [%]")

currentFigure = currentFigure+1;
```

```
figureNames(currentFigure) = "2sigmaarea_mask.eps";
figureTitle = "Mean 2*sigma clustering (described by ellipse) (b)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize.*[2,1]]);
hold on
grid on
plot(sampleSpan,ellipseMeanArea(4:end,:)/ellipseMaxArea*100)
xline(timeSpan(constants.SIGNALFRAME),'-',{'Signal shown'})
xline(timeSpan(constants.STOPPEDFRAME),'-',{'Car stopped'})
legend(conditions.MASK,"Location","southeast")
xlim([timeSpan(1),timeSpan(end)])
ylim([0,20])
xlabel("Elapsed time [s]")
ylabel("Ellipse size [%]")

end
```

Listing 24: PlotEllipseSize.m

```
function [figureAxes,figureNames] = ...
    PlotEccentricity( ...
        sampleSpan, ...
        timeSpan, ...
        eccentricityMean, ...
        screenSize, ...
        figureSize, ...
        conditions, ...
        constants)

currentFigure = 1;
figureNames(currentFigure) = "eccentricity_ehmi.eps";
figureTitle = "Mean ellipse eccentricity (a)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize.*[2,1]]);
hold on
grid on
plot(sampleSpan,eccentricityMean(1:3,:)*100)
xline(timeSpan(constants.SIGNALFRAME),'-',{'Signal shown'})
xline(timeSpan(constants.STOPPEDFRAME),'-',{'Car stopped'})
legend(conditions.EHMI,"Location","southeast")
xlim([timeSpan(1),timeSpan(end)])
ylim([0,100])
xlabel("Elapsed time [s]")
ylabel("Eccentricity [%]")

currentFigure = currentFigure+1;
figureNames(currentFigure) = "eccentricity_mask.eps";
figureTitle = "Mean ellipse eccentricity (b)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize.*[2,1]]);
hold on
grid on
plot(sampleSpan,eccentricityMean(4:end,:)*100)
xline(timeSpan(constants.SIGNALFRAME),'-',{'Signal shown'})
xline(timeSpan(constants.STOPPEDFRAME),'-',{'Car stopped'})
legend(conditions.MASK,"Location","southeast")
xlim([timeSpan(1),timeSpan(end)])
ylim([0,100])
xlabel("Elapsed time [s]")
ylabel("Eccentricity [%]")
```

```

end
```

Listing 25: PlotEccentricity.m

```
function [figureAxes,figureNames] = ...
    PlotBoxCharts( ...
        ehmiGroup, ...
        maskGroup, ...
        ehmiResponseTimes, ...
        ehmiResponseTimeMeans, ...
        maskResponseTimes, ...
        maskResponseTimeMeans, ...
        ehmiSaccadeCounts, ...
        ehmiSaccadeCountsMeans, ...
        maskSaccadeCounts, ...
        maskSaccadeCountsMeans, ...
        ehmiSaccadeAverageAmplitudes, ...
        ehmiSaccadeAverageAmplitudesMeans, ...
        maskSaccadeAverageAmplitudes, ...
        maskSaccadeAverageAmplitudesMeans, ...
        ehmiFixationAverageDurations, ...
        ehmiFixationAverageDurationsMeans, ...
        maskFixationAverageDurations, ...
        maskFixationAverageDurationsMeans, ...
        screenSize, ...
        figureSize)

currentFigure = 1;
figureNames(currentFigure) = "boxchart1a.eps";
figureTitle = "Mean response time per group, measured from "+ ...
    "t_0 = t_{signal} (a)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize]);
hold on
boxchart(ehmiGroup,ehmiResponseTimes,"Notch","on")
plot(ehmiResponseTimeMeans,'b','LineStyle','none','Marker','.')
ylim([0,4.5])
xlabel("Signal type")
ylabel("Response time [s]")

currentFigure = currentFigure+1;
figureNames(currentFigure) = "boxchart1b.eps";
figureTitle = "Mean response time per group, measured from "+ ...
    "t_0 = t_{signal} (b)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize]);
hold on
boxchart(maskGroup,maskResponseTimes,"Notch","on")
plot(maskResponseTimeMeans,'b','LineStyle','none','Marker','.')
ylim([0,4.5])
xlabel("Gaze window size")
ylabel("Response time [s]")

currentFigure = currentFigure+1;
figureNames(currentFigure) = "boxchart2a.eps";
figureTitle = "Saccade/fixation count per group, measured over "+ ...
    "the whole duration (a)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize]);
```

```
hold on
boxchart(ehmiGroup,ehmiSaccadeCounts,"Notch","on")
plot(ehmiSaccadeCountsMeans,'b','LineStyle','none','Marker','.')
ylim([0,30])
xlabel("Signal type")
ylabel("Count [-]")

currentFigure = currentFigure+1;
figureNames(currentFigure) = "boxchart2b.eps";
figureTitle = "Saccade/fixation count per group, measured over "+ ...
    "the whole duration (b)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize]);
hold on
boxchart(maskGroup,maskSaccadeCounts,"Notch","on")
plot(maskSaccadeCountsMeans,'b','LineStyle','none','Marker','.')
ylim([0,30])
xlabel("Gaze window size")
ylabel("Count [-]")

currentFigure = currentFigure+1;
figureNames(currentFigure) = "boxchart3a.eps";
figureTitle = "Average saccade amplitude per group, measured over "+...
    "the whole duration (a)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize]);
hold on
boxchart(ehmiGroup,ehmiSaccadeAverageAmplitudes,"Notch","on")
plot(ehmiSaccadeAverageAmplitudesMeans,'b','LineStyle','none','Marker','.')
ylim([100,400])
xlabel("Signal type")
ylabel("Amplitude [px]")

currentFigure = currentFigure+1;
figureNames(currentFigure) = "boxchart3b.eps";
figureTitle = "Average saccade amplitude per group, measured over "+ ...
    "the whole duration (b)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize]);
hold on
boxchart(maskGroup,maskSaccadeAverageAmplitudes,"Notch","on")
plot(maskSaccadeAverageAmplitudesMeans,'b','LineStyle','none','Marker','.')
ylim([100,400])
xlabel("Gaze window size")
ylabel("Amplitude [px]")

currentFigure = currentFigure+1;
figureNames(currentFigure) = "boxchart4a.eps";
figureTitle = "Average fixation duration per group, measured over "+ ...
    "the whole duration (a)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize]);
hold on
boxchart(ehmiGroup,ehmiFixationAverageDurations,"Notch","on")
plot(ehmiFixationAverageDurationsMeans,'b','LineStyle','none','Marker','.')
ylim([300,1100])
xlabel("Signal type")
ylabel("Duration [ms]")

currentFigure = currentFigure+1;
figureNames(currentFigure) = "boxchart4b.eps";
```

```
figureTitle = "Average fixation duration per group, measured over "+ ...
    "the whole duration (b)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize]);
hold on
boxchart(maskGroup,maskFixationAverageDurations,"Notch","on")
plot(maskFixationAverageDurationsMeans,'b','LineStyle','none','Marker','.')
ylim([300,1100])
xlabel("Gaze window size")
ylabel("Duration [ms]")

end
```

Listing 26: `PlotBoxCharts.m`

```
function [figureAxes,figureNames] = ...
    PlotScatterPlots( ...
        ehmiGroup, ...
        maskGroup, ...
        ehmiResponseTimes, ...
        maskResponseTimes, ...
        ehmiSaccadeCounts, ...
        maskSaccadeCounts, ...
        ehmiSaccadeAverageAmplitudes, ...
        maskSaccadeAverageAmplitudes, ...
        ehmiFixationAverageDurations, ...
        maskFixationAverageDurations, ...
        screenSize, ...
        figureSize, ...
        conditions)

sep = [0,10000];

currentFigure = 1;
figureNames(currentFigure) = "scatter1a.eps";
figureTitle = "Mean response time of experimental groups against "+ ...
    "control group, measured from t_0 = t_{signal} (a)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize]);
hold on
grid on
scatter(ehmiResponseTimes(ehmiGroup==string(conditions.EHMI(3))), ...
    [ehmiResponseTimes(ehmiGroup==string(conditions.EHMI(1))), ...
    ehmiResponseTimes(ehmiGroup==string(conditions.EHMI(2)))], ...
    "filled")
plot(sep,sep,'k--')
legend(conditions.EHMI,"Location","southeast")
xlim([min([ehmiResponseTimes;maskResponseTimes]), ...
    max([ehmiResponseTimes;maskResponseTimes])])
ylim([min([ehmiResponseTimes;maskResponseTimes]), ...
    max([ehmiResponseTimes;maskResponseTimes])])
xlabel("Response time [s] (control group)")
ylabel("Response time [s] (experimental groups)")

currentFigure = currentFigure+1;
figureNames(currentFigure) = "scatter1b.eps";
figureTitle = "Mean response time of experimental groups against "+ ...
    "control group, measured from t_0 = t_{signal} (b)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize]);
```

```
hold on
grid on
scatter(maskResponseTimes(maskGroup==string(conditions.MASK(3))), ...
    [maskResponseTimes(maskGroup==string(conditions.MASK(1))), ...
    maskResponseTimes(maskGroup==string(conditions.MASK(2)))], ...
    "filled")
plot(sep,sep,'k--')
legend(conditions.MASK,"Location","southeast")
xlim([min([ehmiResponseTimes;maskResponseTimes]), ...
    max([ehmiResponseTimes;maskResponseTimes])])
ylim([min([ehmiResponseTimes;maskResponseTimes]), ...
    max([ehmiResponseTimes;maskResponseTimes])])
xlabel("Response time [s] (control group)")
ylabel("Response time [s] (experimental groups)")

currentFigure = currentFigure+1;
figureNames(currentFigure) = "scatter2a.eps";
figureTitle = "Saccade/fixation count of experimental groups against "+ ...
    "control group, measured over the whole duration (a)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize]);
hold on
grid on
scatter(ehmiSaccadeCounts(ehmiGroup==string(conditions.EHMI(3))), ...
    [ehmiSaccadeCounts(ehmiGroup==string(conditions.EHMI(1))), ...
    ehmiSaccadeCounts(ehmiGroup==string(conditions.EHMI(2)))], ...
    "filled")
plot(sep,sep,'k--')
legend(conditions.EHMI,"Location","southeast")
xlim([min([ehmiSaccadeCounts;maskSaccadeCounts]), ...
    max([ehmiSaccadeCounts;maskSaccadeCounts])])
ylim([min([ehmiSaccadeCounts;maskSaccadeCounts]), ...
    max([ehmiSaccadeCounts;maskSaccadeCounts])])
xlabel("Saccade count [-] (control group)")
ylabel("Saccade count [-] (experimental groups)")

currentFigure = currentFigure+1;
figureNames(currentFigure) = "scatter2b.eps";
figureTitle = "Saccade/fixation count of experimental groups against "+ ...
    "control group, measured over the whole duration (b)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize]);
hold on
grid on
scatter(maskSaccadeCounts(maskGroup==string(conditions.MASK(3))), ...
    [maskSaccadeCounts(maskGroup==string(conditions.MASK(1))), ...
    maskSaccadeCounts(maskGroup==string(conditions.MASK(2)))], ...
    "filled")
plot(sep,sep,'k--')
legend(conditions.MASK,"Location","southeast")
xlim([min([ehmiSaccadeCounts;maskSaccadeCounts]), ...
    max([ehmiSaccadeCounts;maskSaccadeCounts])])
ylim([min([ehmiSaccadeCounts;maskSaccadeCounts]), ...
    max([ehmiSaccadeCounts;maskSaccadeCounts])])
xlabel("Saccade count [-] (control group)")
ylabel("Saccade count [-] (experimental groups)")

currentFigure = currentFigure+1;
figureNames(currentFigure) = "scatter3a.eps";
figureTitle = "Average saccade amplitude of experimental groups "+ ...
    "against control group, measured over the whole duration (a)";
```

```
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize]);
hold on
grid on
scatter(ehmiSaccadeAverageAmplitudes( ...
        ehmiGroup==string(conditions.EHMI(3))), ...
    [ehmiSaccadeAverageAmplitudes( ...
        ehmiGroup==string(conditions.EHMI(1))), ...
    ehmiSaccadeAverageAmplitudes( ...
        ehmiGroup==string(conditions.EHMI(2)))], ...
    "filled")
plot(sep,sep,'k--')
legend(conditions.EHMI,"Location","southeast")
xlim([min([ehmiSaccadeAverageAmplitudes;maskSaccadeAverageAmplitudes]), ...
    max([ehmiSaccadeAverageAmplitudes;maskSaccadeAverageAmplitudes])])
ylim([min([ehmiSaccadeAverageAmplitudes;maskSaccadeAverageAmplitudes]), ...
    max([ehmiSaccadeAverageAmplitudes;maskSaccadeAverageAmplitudes])])
xlabel("Amplitude [px] (control group)")
ylabel("Amplitude [px] (experimental groups)")

currentFigure = currentFigure+1;
figureNames(currentFigure) = "scatter3b.eps";
figureTitle = "Average saccade amplitude of experimental groups "+ ...
    "against control group, measured over the whole duration (b)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize]);
hold on
grid on
scatter(maskSaccadeAverageAmplitudes( ...
        maskGroup==string(conditions.MASK(3))), ...
    [maskSaccadeAverageAmplitudes( ...
        maskGroup==string(conditions.MASK(1))), ...
    maskSaccadeAverageAmplitudes( ...
        maskGroup==string(conditions.MASK(2)))], ...
    "filled")
plot(sep,sep,'k--')
legend(conditions.MASK,"Location","southeast")
xlim([min([ehmiSaccadeAverageAmplitudes;maskSaccadeAverageAmplitudes]), ...
    max([ehmiSaccadeAverageAmplitudes;maskSaccadeAverageAmplitudes])])
ylim([min([ehmiSaccadeAverageAmplitudes;maskSaccadeAverageAmplitudes]), ...
    max([ehmiSaccadeAverageAmplitudes;maskSaccadeAverageAmplitudes])])
xlabel("Amplitude [px] (control group)")
ylabel("Amplitude [px] (experimental groups)")

currentFigure = currentFigure+1;
figureNames(currentFigure) = "scatter4a.eps";
figureTitle = "Average fixation duration of experimental groups "+ ...
    "against control group, measured over the whole duration (a)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize]);
hold on
grid on
scatter(ehmiFixationAverageDurations( ...
        ehmiGroup==string(conditions.EHMI(3))), ...
    [ehmiFixationAverageDurations( ...
        ehmiGroup==string(conditions.EHMI(1))), ...
    ehmiFixationAverageDurations( ...
        ehmiGroup==string(conditions.EHMI(2)))], ...
    "filled")
plot(sep,sep,'k--')
legend(conditions.EHMI,"Location","southeast")
```

```
xlim([min([ehmiFixationAverageDurations;maskFixationAverageDurations]), ...
    max([ehmiFixationAverageDurations;maskFixationAverageDurations])])
ylim([min([ehmiFixationAverageDurations;maskFixationAverageDurations]), ...
    max([ehmiFixationAverageDurations;maskFixationAverageDurations])])
xlabel("Duration [ms] (control group)")
ylabel("Duration [ms] (experimental groups)")

currentFigure = currentFigure+1;
figureNames(currentFigure) = "scatter4b.eps";
figureTitle = "Average fixation duration of experimental groups "+ ...
    "against control group, measured over the whole duration (b)";
figureAxes(currentFigure) = figure("Name",figureTitle, ...
    "Position",[screenSize/3,figureSize]);
hold on
grid on
scatter(maskFixationAverageDurations( ...
        maskGroup==string(conditions.MASK(3))), ...
    [maskFixationAverageDurations( ...
        maskGroup==string(conditions.MASK(1))), ...
    maskFixationAverageDurations( ...
        maskGroup==string(conditions.MASK(2)))], ...
    "filled")
plot(sep,sep,'k--')
legend(conditions.MASK,"Location","southeast")
xlim([min([ehmiFixationAverageDurations;maskFixationAverageDurations]), ...
    max([ehmiFixationAverageDurations;maskFixationAverageDurations])])
ylim([min([ehmiFixationAverageDurations;maskFixationAverageDurations]), ...
    max([ehmiFixationAverageDurations;maskFixationAverageDurations])])
xlabel("Duration [ms] (control group)")
ylabel("Duration [ms] (experimental groups)")

end
```

Listing 27: PlotScatterPlots.m

```
function [gazeXNew, ...
        gazeYNew, ...
        pupilSizeNew, ...
        meanXNew, ...
        meanYNew, ...
        stdXPrimeNew, ...
        stdYPrimeNew, ...
        thetaNew] = ...
    InterpolateData( ...
        gazeXOld, ...
        gazeYOld, ...
        pupilSizeOld, ...
        meanXOld, ...
        meanYOld, ...
        stdXPrimeOld, ...
        stdYPrimeOld, ...
        thetaOld, ...
        sampleSpan, ...
        timeSpan, ...
        constants)

% This function converts the arrays sampled at the frequency of the EyeLink
% (2000Hz) to arrays sampled at the framerate of the stimuli (60fps), using
% interpolation.
%
```

```
% Inputs
% - gazeXOld, gazeYOld, pupilSizeOld are nxm cell matrices, where n equals
%   the number of trials and m equals the number of participants. Each cell
%   contains an ix1 array, where i equals the number of data points sampled
%   at the frequency of the EyeLink. They contain the gaze data for each
%   trial of each of the participants.
% - meanXOld, meanYOld, stdXPrimeOld, stdYPrimeOld, thetaOld are nxi
%   arrays, where n equals the number of trials and i equals the number of
%   data points sampled at the frequency of the EyeLink. They contain the
%   dispersion data necessary to draw the deviational ellipse.
% - sampleSpan is an 1xi array, where i equals the number of data points
%   sampled at the frequency of the EyeLink. It contains the time stamps of
%   each of the samples at this frequency.
% - timeSpan is a 1xj array, where j equals the number of data points
%   sampled at the framerate of the stimuli. It contains the time stamps of
%   each of the samples at this frequency.
% - constants is a struct containing various global constants.
%
% Outputs
% - gazeXNew, gazeYNew, pupilSizeNew are nxm cell matrices, where n equals
%   the number of trials and m equals the number of participants. Each cell
%   contains a jx1 array, where j equals the number of data points sampled
%   at the framerate of the stimuli. They contain the gaze data for each
%   trial of each of the participants.
% - meanXNew, meanYNew, stdXPrimeNew, stdYPrimeNew, thetaNew are nxj
%   arrays, where n equals the number of trials and j equals the number of
%   data points sampled at the framerate of the stimuli. They contain the
%   dispersion data necessary to draw the deviational ellipse.

disp("Interpolating processed data.")

tic

[gazeXNew,gazeYNew,pupilSizeNew] = ...
    deal(cell(constants.NUMTRIALS,constants.NUMPARTICIPANTS));
[meanXNew,meanYNew,stdXPrimeNew,stdYPrimeNew,thetaNew] = ...
    deal(nan(constants.NUMTRIALS,numel(timeSpan)));

for trial = 1:constants.NUMTRIALS
    for participant = 1:constants.NUMPARTICIPANTS
        gazeXNew{trial,participant} = ...
            interp1(sampleSpan,gazeXOld{trial,participant},timeSpan)';
        gazeYNew{trial,participant} = ...
            interp1(sampleSpan,gazeYOld{trial,participant},timeSpan)';
        pupilSizeNew{trial,participant} = ...
            interp1(sampleSpan,pupilSizeOld{trial,participant},timeSpan)';
    end
    meanXNew(trial,:) = ...
        interp1(sampleSpan,meanXOld(trial,:),timeSpan);
    meanYNew(trial,:) = ...
        interp1(sampleSpan,meanYOld(trial,:),timeSpan);
    stdXPrimeNew(trial,:) = ...
        interp1(sampleSpan,stdXPrimeOld(trial,:),timeSpan);
    stdYPrimeNew(trial,:) = ...
        interp1(sampleSpan,stdYPrimeOld(trial,:),timeSpan);
    thetaNew(trial,:) = ...
        interp1(sampleSpan,thetaOld(trial,:),timeSpan);
end
toc

```

```
end
```

Listing 28: InterpolateData.m

```
function keyPressesSampled = ...
    SampleKeyPresses( ...
        timeSpan, ...
        keyPresses, ...
        constants)

% This function transforms the event-based key press data into a
% time-sampled version.
%
% Inputs
% - timeSpan is a 1xj array, where j equals the number of data points
%   sampled at the framerate of the stimuli. It contains the time stamps of
%   each of the samples at this frequency.
% - keyPresses is an nxm cell matrix, where n is the number of trials and m
%   is the number of participants. Each cell contains the key press events
%   for that specific trial of that specific participant.
% - constants is a struct containing various global constants.
%
% Outputs
% - keyPressesSampled is an nxm cell matrix, where n is the number of
%   trials and m is the number of participants. Each cell contains a 1xj
%   array, where j is the number of data points sampled at the framerate of
%   the stimuli. They are 0 when a key has not been pressed, and 1 when a
%   key has been pressed.

keyPressesSampled = ...
    deal(cell(constants.NUMTRIALS,constants.NUMPARTICIPANTS));
for trial = 1:constants.NUMTRIALS
    for participant = 1:constants.NUMPARTICIPANTS
        keyPressesSampled{trial,participant} = zeros(size(timeSpan));
        for keyPress = 1:size(keyPresses{trial,participant},1)
            if keyPresses{trial,participant}(keyPress,1)<0
                continue
            elseif ~mod(keyPress,2)
                keyPressesSampled{trial,participant}(find( ...
                    timeSpan>keyPresses{trial,participant}( ...
                        keyPress,1),1):end) = constants.KEYPRESSED;
            end
        end
    end
end

end
```

Listing 29: SampleKeyPresses.m

```
function DrawGazeOnStimulus( ...
    trial, ...
    gazeX, ...
    gazeY, ...
    meanX, ...
    meanY, ...
    stdXPrime, ...
    stdYPrime, ...
    theta, ...
    scaleFactor, ...
    trialTable, ...
    keyPresses, ...
    drawEllipse, ...
    stimulusGreyScale, ...
    constants)

% This function draws the gaze positions of all the participants as markers
% on the original stimulus. The markers have a color indication to indicate
% whether or not they performed the key press for that trial. Additionally,
% there is an option to draw the deviational ellipse as well. The stimuli
% files need to be present in the "stimuli" subdirectory relative to the
% MATLAB working directory, with the following filename formatting:
% "preset_<preset>_ehmi_<ehmi>.mp4", where <preset> contains the preset
% number, and <ehmi> contains the ehmi condition. The output files will be
% written to the "stimuli_gaze_overlay" subdirectory, relative to the
% MATLAB working directory.
%
% Inputs
% - trial, the trial number the markers will be written for.
% - gazeX, gazeY are nxm cells where n equals the number of trials, and m
%   equals the number of participants. They contain the participants' gaze
%   data. The number of data points must match the number of frames in the
%   trial.
% - meanX, meanY, stdXPrime, stdYPrime, theta are nxi matrices, where n
%   equals the number of trials and i equals the number of data points.
%   They contain the dispersion data necessary to draw the deviational
%   ellipse. The number of data points must match the number of frames in
%   the trial.
% - scaleFactor is a 1x1 double containing the scaleFactor used to scale
%   scale the deviational ellipse to the 95% confidence interval.
% - trialTable is an nx3 table, containing the preset number and
%   independent variables for all n trials.
% - keyPresses is an nxm cell, where n equals the number of trials, and m
%   equals the number of participants. It contains sampled key press data
%   indicating for each frame of each trial whether or not a participant
%   has pressed the key.
% - drawEllipse is a boolean. Draws the derivational ellipse if set to
%   true. Does not draw the derivational ellipse if not set to true.
% - stimulusGreyScale is a boolean. Coverts the stimulus to a greyscale
%   version if set to true. The stimulus remains full color if not set to
%   true.
% - constants is a struct containing various global constants.

disp(strcat("Drawing gaze data on stimulus ",string(trial), ...
    "/",string(size(trialTable,1)),"."))

tic

inputFile = strcat("stimuli\preset_",string(trialTable.PRESET(trial)), ...
    "_ehmi_",lower(string(trialTable.EHMI(trial))),".mp4");
outputFile = strcat("stimuli_gaze_overlay\stimulus_",string(trial));

currentStimulus = VideoReader(inputFile);
currentStimulus.CurrentTime = currentStimulus.Duration-constants.DURATION;

% Output to uncompressed avi, encode afterwards with ffmpeg.
videoContainer = VideoWriter(outputFile,"Uncompressed AVI");
videoContainer.FrameRate = 60;

open(videoContainer);
```

```matlab

markerSize = 3;
phi = linspace(0,2*pi,300);
for frame = 1:constants.FRAMERATE*constants.DURATION
    currentVideoFrame = readFrame(currentStimulus);
    if stimulusGreyScale
        currentVideoFrame = rgb2gray(currentVideoFrame);
    end

    % Insert gaze markers
    for participant = 1:constants.NUMPARTICIPANTS
        if isfinite([gazeX{trial,participant}(frame), ...
                gazeY{trial,participant}(frame)])
            if keyPresses{trial,participant}(frame) == 1
                markerColor = "green";
            else
                markerColor = "yellow";
            end
            currentVideoFrame = insertMarker(currentVideoFrame, ...
                [gazeX{trial,participant}(frame), ...
                gazeY{trial,participant}(frame)], ...
                "Color",markerColor,"Size",markerSize);
        end
    end

    % Calculate and insert dispersion ellipse if necessary
    if drawEllipse
        % use inverse ROT(th) = [cos(th) -sin(th), sin(th), cos(th)]^-1
        ellipseX = meanX(trial,frame)+ ...
            scaleFactor*stdXPrime(trial,frame)* ...
            sin(phi)*cos(theta(trial,frame))+ ...
            scaleFactor*stdYPrime(trial,frame)* ...
            cos(phi)*sin(theta(trial,frame));
        ellipseY = meanY(trial,frame)- ...
            scaleFactor*stdXPrime(trial,frame)* ...
            sin(phi)*sin(theta(trial,frame))+ ...
            scaleFactor*stdYPrime(trial,frame)* ...
            cos(phi)*cos(theta(trial,frame));

        if isfinite([ellipseX,ellipseY])
            currentVideoFrame = insertShape(currentVideoFrame, ...
                "Polygon",[ellipseX',ellipseY'], ...
                "LineWidth",1,"Color","yellow");
        end
    end

    % Insert text overlay
    currentVideoFrame = insertText(currentVideoFrame,[10,10], ...
        strcat("Stimulus:",string(trial), ...
        " (Preset:",string(trialTable.PRESET(trial)), ...
        ",Signal:",lower(string(trialTable.EHMI(trial))), ...
        ",Gaze:",lower(string(trialTable.MASK(trial))), ...
        "), Frame: ",string(frame)), ...
        "FontSize",18,"BoxColor","black", ...
        "BoxOpacity",0.5,"TextColor","white");

    writeVideo(videoContainer,currentVideoFrame);
end

close(videoContainer)

```

```
% Encode uncompressed avi to x264.
system(sprintf(['ffmpeg -hide_banner -loglevel error -y -i %s.avi ' ...
    '-preset slower -pix_fmt yuv420p -profile:v high -level:v 5.0 ' ...
    '-crf 18 ' ...
    '-x264-params bframes=0:keyint=30:min-keyint=3:rc-lookahead=30 ' ...
    '%s.mp4'], ...
    outputFile,outputFile));
delete(strcat(outputFile,".avi"))

toc

end
```

Listing 30: DrawGazeOnStimulus.m

```
function ReconstructTrial( ...
    trial, ...
    participant, ...
    gazeX, ...
    gazeY, ...
    trialTable, ...
    constants)

% This function recreates a specific trial from a specific participant, by
% drawinging the gaze contingent window on the original stimulus, using the
% gaze data for that trial. The stimuli files need to be present in the
% "stimuli" subdirectory relative to the MATLAB working directory, with the
% following filename formatting: "preset_<preset>_ehmi_<ehmi>.mp4", where
% <preset> contains the preset number, and <ehmi> contains the ehmi
% condition. The blurred versions of the same stimuli files need to be
% present in the "stimuli_blur" subdirectory relative to the MATLAB working
% directory, with the following filename formatting:
% "preset_<preset>_ehmi_<ehmi>.mp4", where <preset> contains the preset
% number, and <ehmi> contains the ehmi condition. Additionally it will need
% the appropriate mask files used in the experiment in the "mask"
% subdirectory relative to the MATLAB working directory, with the following
% filename formatting: "<size>.png", where <size> is the name of the gaze
% window condition. The output files will be written to the "stimuli_gaze"
% subdirectory, relative to the MATLAB working directory.
%
% Inputs
% - trial is an int, specifying the trial to be recreated.
% - participant is an int, specifying the participant of that trial.
% - gazeX, gazeY are nxm cell matrices, where n equals the number of trials
%   and m equals the number of participants. Each cell contains an jx1
%   array, where j equals the number of data points sampled at the
%   framerate of the stimulus. They contain the gaze data for each trial of
%   each of the participants.
% - trialTable is a nxm table, containing the desired order of the trials,
%   where n is the number of trials, and m is the number of identifier
%   categories.
% - constants is a struct containing various global constants.

disp(strcat("Reconstructing trial performance ", ...
    string(trial),"/",string(size(trialTable,1)), ...
    " for participant ",string(participant),"."))

tic
inputFileClear = strcat("stimuli\preset_", ...
    string(trialTable.PRESET(trial)),"_ehmi_", ...
    lower(string(trialTable.EHMI(trial))),".mp4");
inputFileBlur = strcat("stimuli_blur\preset_", ...
    string(trialTable.PRESET(trial)),"_ehmi_", ...
    lower(string(trialTable.EHMI(trial))),".mp4");
smallMask = imread(strcat("mask\", ...
    lower(string(trialTable.MASK(trial))),".png"));
smallMask = ~(im2double(smallMask(:,:,3)));

outputFile = strcat("stimuli_gaze\stimulus_",string(trial));

currentStimulusClear = VideoReader(inputFileClear);
currentStimulusClear.CurrentTime = ...
    currentStimulusClear.Duration-constants.DURATION;

currentStimulusBlur = VideoReader(inputFileBlur);
currentStimulusBlur.CurrentTime = ...
    currentStimulusBlur.Duration-constants.DURATION;

% Output to uncompressed avi, encode afterwards with ffmpeg.
videoContainer = VideoWriter(outputFile,"Uncompressed AVI");
videoContainer.FrameRate = 60;

open(videoContainer);

for frame = 1:constants.FRAMERATE*constants.DURATION
    currentVideoFrameClear = readFrame(currentStimulusClear);
    currentVideoFrameBlur = readFrame(currentStimulusBlur);

    bigMask = zeros(constants.SCREENHEIGHT,constants.SCREENWIDTH);

    % Perform checks and apply mask cropped to screen size
    if trialTable.MASK(trial) == "NONE"
        currentVideoFrameCombined = currentVideoFrameClear;
    elseif isfinite([gazeX{trial,participant}(frame), ...
            gazeY{trial,participant}(frame)])
        cropMask = smallMask;
        left = floor(gazeX{trial,participant}(frame)- ...
            size(smallMask,1)/2);
        right = floor(gazeX{trial,participant}(frame)+ ...
            size(smallMask,1)/2)-1;
        up = floor(gazeY{trial,participant}(frame)- ...
            size(smallMask,2)/2);
        down = floor(gazeY{trial,participant}(frame)+ ...
            size(smallMask,2)/2)-1;

        if right>constants.SCREENWIDTH
            cropMask(:,constants.SCREENWIDTH+end-right+1:end) = [];
            right = constants.SCREENWIDTH;
        end
        if left<1
            cropMask(:,1:1-left) = [];
            left = 1;
        end
        if down>constants.SCREENHEIGHT
            cropMask(constants.SCREENHEIGHT+end-down+1:end,:) = [];
            down = constants.SCREENHEIGHT;
        end
        if up<1
            cropMask(1:1-up,:) = [];
            up = 1;
        end

        bigMask(up:down,left:right) = cropMask;

        blender = vision.AlphaBlender("Operation","Binary Mask", ...
            "MaskSource","Input port");

        currentVideoFrameCombined = blender(currentVideoFrameBlur, ...
            currentVideoFrameClear,bigMask);
    else
        currentVideoFrameCombined = currentVideoFrameBlur;
    end

    % Insert text overlay
    currentVideoFrameCombined = ...
        insertText(currentVideoFrameCombined,[10,10], ...
        strcat("Stimulus:",string(trial), ...
        " (Preset:",string(trialTable.PRESET(trial)), ...
        ",Type:",lower(string(trialTable.EHMI(trial))), ...
        ",Mask:",lower(string(trialTable.MASK(trial))), ...
        "), Frame: ",string(frame)), ...
        "FontSize",18,"BoxColor","black", ...
        "BoxOpacity",0.5,"TextColor","white");

    writeVideo(videoContainer,currentVideoFrameCombined);
end

close(videoContainer)

% Encode uncompressed avi to x264.
system(sprintf(['ffmpeg -hide_banner -loglevel error -y -i %s.avi ' ...
    '-preset slower -pix_fmt yuv420p -profile:v high -level:v 5.0 ' ...
    '-crf 18 ' ...
    '-x264-params bframes=0:keyint=30:min-keyint=3:rc-lookahead=30 ' ...
    '%s.mp4'],outputFile,outputFile));
delete(strcat(outputFile,".avi"))

toc

end
```

Listing 31: ReconstructTrial.m

C.1.3. MISCELLANEOUS

```
% Reproduce car profiles defined in Blender python.
clc
clear
close all

%% Initialize
figDir = "figures";

profileDur = 35; % [s]
frameRate = 60; % [1/s]
numFrames = frameRate*profileDur; % [-]
deltaT = 1/frameRate; % [s]

numCars = 4; % [-]
numPresets = 7; % [-]

```

```
lenRoutes = [114.62288, 91.11199, 106.71832]; % [m, m, m], from blender
distOriginToRouteEnd = 15; % [m]
distOriginToStopTarget = -5; % [m]

accInit = 3; % [m/s^2]
accStop = -accInit; % [m/s^2]
velInit = 0; % [m/s]
velTarget = 30*1000/(60*60); % [m/s] (30 km/h)
distInit = 0; % [m]

distBraking = -velTarget^2 / (2*accStop); % [s]

stimDur = 10; % [s]
stimNumFrames = frameRate*stimDur; % [s]

maxError = 1e-10;

%% Iterate over all frames for all profiles
% Initialize arrays for results and values of interest.
[dist,vel,acc] = deal(nan(numFrames,numCars,numPresets));
[signalFrame,carStopped] = deal(zeros(numPresets,1));
firstToStop = ones(numPresets,1)*(numCars+1);

% Iterate over all presets.
for preset = 1:numPresets
    distVecs = nan(numFrames,numCars);
    velVecs = nan(numFrames,numCars);
    accVecs = nan(numFrames,numCars);

    % Define presets.
    if preset == 1
        delaysT = [15/6, 5/6, 0, 15/6];
        routeID = [2, 1, 0, 0] + 1;
        stopping = [false, false, false, true];
    elseif preset == 2
        delaysT = [35/6, 0, 30/6, 25/6];
        routeID = [2, 0, 1, 0] + 1;
        stopping = [false, false, false, true];
    elseif preset == 3
        delaysT = [0, 15/6, 45/6, 65/6];
        routeID = [0, 0, 1, 2] + 1;
        stopping = [false, false, true, false];
    elseif preset == 4
        delaysT = [5/6, 0, 30/6, 25/6];
        routeID = [1, 0, 1, 0] + 1;
        stopping = [false, false, false, true];
    elseif preset == 5
        delaysT = [5/6, 0, 30/6, 25/6];
        routeID = [1, 0, 1, 0] + 1;
        stopping = [false, false, true, true];
    elseif preset == 6
        delaysT = [15/6, 5/6, 0, 15/6];
        routeID = [2, 1, 0, 0] + 1;
        stopping = [false, false, true, true];
    elseif preset == 7
        delaysT = [15/6, 5/6, 0, 15/6];
        routeID = [2, 1, 0, 0] + 1;
        stopping = [false, false, false, false];
    end

    stopID = 0;

    % If multiple cars are stopping, next one is stopping behind the
    % previous one.
    distTarget = lenRoutes(routeID);
    for i = 1:numel(stopping)
        if stopping(i)
            if firstToStop(preset) == numCars+1
                firstToStop(preset) = i;
            end
            distTarget(i) = distTarget(i) - ...
                (distOriginToRouteEnd - distOriginToStopTarget) - ...
                7*stopID;
            stopID = stopID + 1;
        end
    end

    % Pad starting delay with nans.
    for car = 1:numCars
        distVecs(:,car) = [nan(round(delaysT(car)*frameRate),1); ...
            ones(numFrames-round(delaysT(car)*frameRate),1)*distInit];
        velVecs(:,car) = [nan(round(delaysT(car)*frameRate),1); ...
            ones(numFrames-round(delaysT(car)*frameRate),1)*velInit];
        accVecs(:,car) = [nan(round(delaysT(car)*frameRate),1); ...
            ones(numFrames-round(delaysT(car)*frameRate),1)*accInit];
    end

    % Status of a car during the trajectory.
    % state = 0: driving, state = 1: braking, state = 2: stopped.
    state = zeros(1,4);

    % Iterate over all frames in the scene.
    for frame = 2:numFrames
        for car = 1:numCars
            if isnan(distVecs(frame,car)) || ...
                    isnan(distVecs(frame-1,car))
                continue
            end

            % Update car state.
            distVecs(frame,car) = distVecs(frame-1,car) + ...
                velVecs(frame-1,car)*deltaT + ...
                accVecs(frame-1,car)*deltaT^2/2;
            velVecs(frame,car) = velVecs(frame-1,car) + ...
                accVecs(frame-1,car)*deltaT;
            accVecs(frame,car) = accVecs(frame-1,car);

            % Check if target velocity is reached.
            if (velVecs(frame,car) + maxError >= velTarget) && ...
                    ~(accVecs(frame,car) == 0)
                velVecs(frame,car) = velTarget;
                accVecs(frame,car) = 0;
            end

            % Check if car needs to start braking.
            if distVecs(frame,car) + maxError >= ...
                    distTarget(car) - distBraking && ...
                    stopping(car) && ...
                    state(car) == 0
                accVecs(frame,car) = accStop;
                state(car) = state(car) + 1;

```

```matlab
                if car == firstToStop(preset)
                    signalFrame(preset) = frame;
                end
            end

            % Check if car has come to a full stop.
            if velVecs(frame,car) - maxError <= 0 && ...
                    state(car) == 1
                velVecs(frame,car) = 0;
                accVecs(frame,car) = 0;
                state(car) = state(car) + 1;

                if car == firstToStop(preset)
                    carStopped(preset) = frame;
                end
            end
        end
    end

    % Shift distance vectors to cross 0 at the pedestrian's global
    % position.
    for car = 1:numCars
        if routeID(car) < 3
            distVecs(:,car) = distVecs(:,car) - ...
                lenRoutes(routeID(car)) + distOriginToRouteEnd;
        else
            distVecs(:,car) = distVecs(:,car) - ...
                distOriginToRouteEnd;
        end
    end

    dist(:,:,preset) = distVecs;
    vel(:,:,preset) = velVecs;
    acc(:,:,preset) = accVecs;
end

%% Plot
frameVec = linspace(1,numFrames,numFrames);

for preset = 1:numPresets
    if firstToStop(preset) < numCars+1
        stimulusStart = signalFrame(preset) - frameRate * 5 - 2;
        timeVector = [-(stimulusStart-1)/frameRate, ...
            -(stimulusStart-1)/frameRate + frameVec*deltaT] ...
            -(signalFrame(preset)-stimulusStart)/frameRate;
    else
        stimulusStart = frameRate * 8 - 2;
        timeVector = [-(stimulusStart-1)/frameRate, ...
            -(stimulusStart-1)/frameRate + frameVec*deltaT] ...
            -stimDur/2 - (stimDur/2-2)/frameRate;
    end

    figName = strcat("carprofiles",string(preset),".eps");
    figTitle = strcat("Car a-, v- and s-profiles preset ",string(preset));
    currFig = figure("Name",figTitle);
    tiledlayout(3,1,"TileSpacing","compact", ...
        "Padding","compact")

    nexttile
    hold on
    grid on
    plot(timeVector(1:end-1),acc(:,:,preset))
    xline(timeVector(stimulusStart),'-',{'Stimulus start'})
    xline(timeVector(stimulusStart+stimNumFrames),'-',{'Stimulus end'})
    if firstToStop(preset) < numCars+1
        xline(timeVector(signalFrame(preset)),'-',{'Signal shown'})
        xline(timeVector(carStopped(preset)),'-',{'Car stopped'})
    end
    xlim([-3*stimDur/2, 3*stimDur/2])
    ylim([accStop, accInit])
    legend("Car 1","Car 2","Car 3","Car 4","Location","bestoutside")
    xlabel("t [s]")
    ylabel("a [ms^{-2}]")

    nexttile
    hold on
    grid on
    plot(timeVector(1:end-1),vel(:,:,preset))
    xline(timeVector(stimulusStart),'-')
    xline(timeVector(stimulusStart+stimNumFrames),'-')
    if firstToStop(preset) < numCars+1
        xline(timeVector(signalFrame(preset)),'-')
        xline(timeVector(carStopped(preset)),'-')
    end
    xlim([-3*stimDur/2, 3*stimDur/2])
    ylim([velInit, 10])
    xlabel("t [s]")
    ylabel("v [ms^{-1}]")

    nexttile
    hold on
    grid on
    plot(timeVector(1:end-1),dist(:,:,preset))
    xline(timeVector(stimulusStart),'-')
    xline(timeVector(stimulusStart+stimNumFrames),'-')
    if firstToStop(preset) < numCars+1
        xline(timeVector(signalFrame(preset)),'-')
        xline(timeVector(carStopped(preset)),'-')
    end
    xlim([-3*stimDur/2, 3*stimDur/2])
    ylim([-100, 200])
    xlabel("t [s]")
    ylabel("s [m]")

    exportgraphics(currFig,fullfile(figDir,figName), ...
        "ContentType","vector")
end
```

Listing 32: `CarProfiles.m`

```
clc
clear
close all

tic

% Initialize
PRESET = 3;
EHMI = "MESSAGE";

FRAMERATE = 60; % [fps]
```

```matlab
PADDEDDURATION = 1; % [s]
DURATION = 10; % [s]

SIGNALFRAME = 302; % [-]
GAZEX = 850; % [px]
GAZEY = 300; % [px]

SCREENWIDTH = 1280;
SCREENHEIGHT = 720;

maskTypes = ["SMALL","LARGE","NONE"];
frame = PADDEDDURATION*FRAMERATE+SIGNALFRAME;

inputFileClear = strcat("stimuli\preset_", ...
    string(PRESET),"_ehmi_", ...
    lower(EHMI),".mp4");
inputFileBlur = strcat("stimuli_blur\preset_", ...
    string(PRESET),"_ehmi_", ...
    lower(EHMI),".mp4");

currentStimulusClear = VideoReader(inputFileClear);
currentStimulusClear.CurrentTime = frame/FRAMERATE;

currentStimulusBlur = VideoReader(inputFileBlur);
currentStimulusBlur.CurrentTime = frame/FRAMERATE;

currentVideoFrameClear = readFrame(currentStimulusClear);
currentVideoFrameBlur = readFrame(currentStimulusBlur);

% Loop over mask types and output image for specified frame.
for mask = 1:numel(maskTypes)
    smallMask = imread(strcat("mask\", ...
        lower(maskTypes(mask)),".png"));
    smallMask = ~(im2double(smallMask(:,:,3)));

    outputFile = strcat("figures\preset_",string(PRESET), ...
        "_ehmi_",lower(EHMI),"_mask_",lower(maskTypes(mask)),".png");

    bigMask = zeros(SCREENHEIGHT,SCREENWIDTH);

    % Perform checks and apply mask cropped to screen size
    if maskTypes(mask) == "NONE"
        currentVideoFrameCombined = currentVideoFrameClear;
    elseif isfinite([GAZEX,GAZEY])
        cropMask = smallMask;
        left = floor(GAZEX-size(smallMask,1)/2);
        right = floor(GAZEX+size(smallMask,1)/2)-1;
        up = floor(GAZEY-size(smallMask,2)/2);
        down = floor(GAZEY+size(smallMask,2)/2)-1;

        if right>SCREENWIDTH
            cropMask(:,SCREENWIDTH+end-right+1:end) = [];
            right = SCREENWIDTH;
        end
        if left<1
            cropMask(:,1:1-left) = [];
            left = 1;
        end
        if down>SCREENHEIGHT
            cropMask(SCREENHEIGHT+end-down+1:end,:) = [];
            down = SCREENHEIGHT;
        end
        if up<1
            cropMask(1:1-up,:) = [];
            up = 1;
        end

        bigMask(up:down,left:right) = cropMask;

        blender = vision.AlphaBlender("Operation","Binary Mask", ...
            "MaskSource","Input port");

        currentVideoFrameCombined = blender(currentVideoFrameBlur, ...
            currentVideoFrameClear,bigMask);
    else
        currentVideoFrameCombined = currentVideoFrameBlur;
    end

    % Write frame to file
    imwrite(currentVideoFrameCombined,outputFile)
end

toc
```

Listing 33: GazeWindow.m

```
%% Clear Workspace and Command Window
clc
clear
close all

%% Import Excel questionnaire results file
inputSheet = '.\questionnaire\Questionnaire_results.xlsx';

numVars = 7;
varNames = {'Participant','eHMIType', ...
    'Useful','Pleasant','Good','Nice','Effective', ...
    'Likable','Assisting','Desirable','RaisingAlertness', ...
    'Usefulness','Satisfying'} ;
varTypes = {'int32','categorical', ...
    'int32','int32','int32','int32','int32', ...
    'int32','int32','int32','int32', ...
    'double','double'} ;
datRange = 'A2:M70';

opts = spreadsheetImportOptions( ...
    'NumVariables',numVars, ...
    'VariableNames',varNames, ...
    'VariableTypes',varTypes, ...
    'DataRange', datRange);

myTable = readtable(inputSheet,opts);

figureDir = "figures";

%% Transform data
% All the data needs to be in a single column. xgroup vector indicates
% which column the corresponding value originally came from. cgroupdata
% vector indicates which data set the corresponding value came from.

catOrder = {'FLASH','MESSAGE','OFF'};
```

```
orderedCats = categorical(myTable.eHMIType,catOrder,catOrder);

% Usefulness and Satisfying Scale
data1 = [myTable.Usefulness;
    myTable.Satisfying];

xgroup1 = [orderedCats;orderedCats];

cgroup1_ones = ones(size(orderedCats));
cgroup1 = [cgroup1_ones;cgroup1_ones*2];

matrixUsefulness = reshape(myTable.Usefulness,3,[])';
matrixSatisfying = reshape(myTable.Satisfying,3,[])';

meansUsefulness = mean(matrixUsefulness)';
meansSatisfying = mean(matrixSatisfying)';

% Usefulness Scale Components
data2 = double([myTable.Useful;
    myTable.Good;
    myTable.Effective;
    myTable.Assisting;
    myTable.RaisingAlertness]);

xgroup2 = [orderedCats;orderedCats;orderedCats;orderedCats;orderedCats];

cgroup2_ones = ones(size(orderedCats));
cgroup2 = [cgroup2_ones;
    cgroup2_ones*2;
    cgroup2_ones*3;
    cgroup2_ones*4;
    cgroup2_ones*5];

matrixUseful = reshape(myTable.Useful,3,[])';
matrixGood = reshape(myTable.Good,3,[])';
matrixEffective = reshape(myTable.Effective,3,[])';
matrixAssisting = reshape(myTable.Assisting,3,[])';
matrixRaisingAlertness = reshape(myTable.RaisingAlertness,3,[])';

meansUseful = mean(matrixUseful)';
meansGood = mean(matrixGood)';
meansEffective = mean(matrixEffective)';
meansAssisting = mean(matrixAssisting)';
meansRaisingAlertness = mean(matrixRaisingAlertness)';

% Satisfying Scale Components
data3 = double([myTable.Pleasant;
    myTable.Nice;
    myTable.Likable;
    myTable.Desirable]);

xgroup3 = [orderedCats;orderedCats;orderedCats;orderedCats];

cgroup3_ones = ones(size(orderedCats));
cgroup3 = [cgroup3_ones;cgroup3_ones*2;cgroup3_ones*3;cgroup3_ones*4];

matrixPleasant = reshape(myTable.Pleasant,3,[])';
matrixNice = reshape(myTable.Nice,3,[])';
matrixLikable = reshape(myTable.Likable,3,[])';
matrixDesirable = reshape(myTable.Desirable,3,[])';

```

```
meansPleasant = mean(matrixPleasant)';
meansNice = mean(matrixNice)';
meansLikable = mean(matrixLikable)';
meansDesirable = mean(matrixDesirable)';

%% Boxplots
figureName = "questionnaire1.eps";
figureTitle = "Usefulness And Satisfying Scale";
currentFigure = figure("Name",figureTitle);
currentBox = boxchart(xgroup1,data1,'GroupByColor',cgroup1);
currentAxes = currentBox.Parent;
currentAxes.XAxis.Categories = {currentAxes.XAxis.Categories{1}, ...
    'Temp1',currentAxes.XAxis.Categories{2}, ...
    'Temp2',currentAxes.XAxis.Categories{3}};
currentAxes.XTick = [currentAxes.XTick(1), ...
    currentAxes.XTick(3), ...
    currentAxes.XTick(5)];
hold on
plot([0.75,2.75,4.75],meansUsefulness, ...
    'k','LineStyle','none','Marker','.')
plot([1.25,3.25,5.25],meansSatisfying, ...
    'k','LineStyle','none','Marker','.')
legend('Usefulness','Satisfying')

exportgraphics(currentFigure,fullfile(figureDir,figureName), ...
    "ContentType","vector")

figureName = "questionnaire2.eps";
figureTitle = "Usefulness Scale Components";
currentFigure = figure("Name",figureTitle);
currentBox = boxchart(xgroup2,data2,'GroupByColor',cgroup2);
currentAxes = currentBox.Parent;
currentAxes.XAxis.Categories = {currentAxes.XAxis.Categories{1}, ...
    'Temp1',currentAxes.XAxis.Categories{2}, ...
    'Temp2',currentAxes.XAxis.Categories{3}};
currentAxes.XTick = [currentAxes.XTick(1), ...
    currentAxes.XTick(3), ...
    currentAxes.XTick(5)];
hold on
plot([0.6,2.6,4.6],meansUseful, ...
    'k','LineStyle','none','Marker','.')
plot([0.8,2.8,4.8],meansGood, ...
    'k','LineStyle','none','Marker','.')
plot([1,3,5],meansEffective, ...
    'k','LineStyle','none','Marker','.')
plot([1.2,3.2,5.2],meansAssisting, ...
    'k','LineStyle','none','Marker','.')
plot([1.4,3.4,5.4],meansRaisingAlertness, ...
    'k','LineStyle','none','Marker','.')
legend('Useful','Good','Effective','Assisting','Raising Alertness')

exportgraphics(currentFigure,fullfile(figureDir,figureName), ...
    "ContentType","vector")

figureName = "questionnaire3.eps";
figureTitle = "Satisfying Scale Components";
currentFigure = figure("Name",figureTitle);
currentBox = boxchart(xgroup3,data3,'GroupByColor',cgroup3);
currentAxes = currentBox.Parent;
currentAxes.XAxis.Categories = {currentAxes.XAxis.Categories{1}, ...
    'Temp1',currentAxes.XAxis.Categories{2}, ...
    'Temp2',currentAxes.XAxis.Categories{3}};
currentAxes.XTick = [currentAxes.XTick(1), ...
    currentAxes.XTick(3), ...
    currentAxes.XTick(5)];
hold on
plot([0.625,2.625,4.625],meansPleasant, ...
    'k','LineStyle','none','Marker','.')
plot([0.875,2.875,4.875],meansNice, ...
    'k','LineStyle','none','Marker','.')
plot([1.125,3.125,5.125],meansLikable, ...
    'k','LineStyle','none','Marker','.')
plot([1.375,3.375,5.375],meansDesirable, ...
    'k','LineStyle','none','Marker','.')
legend('Pleasant','Nice','Likable','Desirable')

exportgraphics(currentFigure,fullfile(figureDir,figureName), ...
    "ContentType","vector")

%% Radar plots
figureName = "questionnaireradar1.eps";
figureTitle = "Means of usefulness scale components";
currentFigure = figure("Name",figureTitle);
spider_plot_R2019b( ...
    [meansUseful,meansGood,meansEffective,meansAssisting, ...
        meansRaisingAlertness], ...
    "AxesInterval", 4, ...
    "AxesLabels", ...
        {'Useful','Good','Effective','Assisting','Raising Alertness'}, ...
    "AxesLimits", [repmat(-2,1,5); repmat(2,1,5)], ...
    "AxesPrecision", zeros(1,5), ...
    "AxesLabelsEdge", "none", ...
    "Direction", "counterclockwise", ...
    "LineStyle", repmat({'--'},1,3));
legend({'FLASH','MESSAGE','OFF'}, 'Location', 'northeastoutside');

exportgraphics(currentFigure,fullfile(figureDir,figureName), ...
    "ContentType","vector")

figureName = "questionnaireradar2.eps";
figureTitle = "Means of satisfying scale components";
currentFigure = figure("Name",figureTitle);
spider_plot_R2019b( ...
    [meansPleasant,meansNice,meansLikable,meansDesirable], ...
    "AxesInterval", 4, ...
    "AxesLabels", {'Pleasant','Nice','Likable','Desirable'}, ...
    "AxesLimits", [repmat(-2,1,4); repmat(2,1,4)], ...
    "AxesPrecision", zeros(1,4), ...
    "AxesLabelsEdge", "none", ...
    "Direction", "counterclockwise", ...
    "LineStyle", repmat({'--'},1,3));
legend({'FLASH','MESSAGE','OFF'}, 'Location', 'northeastoutside');

exportgraphics(currentFigure,fullfile(figureDir,figureName), ...
    "ContentType","vector")
```

Listing 34: `Questionnaire.m`

```
function mNormalized = NormalizeMatrix(m)

% This function normalizes an entire matrix between its maximum and minimum
% value.
%
% Inputs
% - m is an nxm matrix.
%
% Outputs
% - mNormalized is an nxm matrix.

mMin = min(m(:));
mMax = max(m(:));
mNormalized = (m-mMin)/(mMax-mMin);

end
```

Listing 35: `NormalizeMatrix.m`

C.2. PYTHON

```python
import bpy
import time


class Car:
    """
    This is a class for calculating car velocity profiles to be used in Blender.

    Attributes:
        data_object (str): Object data path.
        a (list):           a at any frame.
        v (list):           v at any frame.
        s (list):           s at any frame.
        c (bool):           Stopping trigger.
        e (bool):           EHMI trigger.
        et (int):           EHMI type (0: text, 1: light).
        es (list):          EHMI toggle at any frame (0: off, 1: on).
        strn (int):         EHMI emission strength.
    """

    def __init__(self, data_object: str, a: list, v: list, s: list, c: bool, e: bool, et:
    ↪   int, es: list, strn: int):
        """
        The constructor of the Car class.

        Parameters:
            data_object (str): Object data path.
            a (list):           a at any frame.
            v (list):           v at any frame.
            s (list):           s at any frame.
            c (bool):           Stopping trigger.
            e (bool):           EHMI trigger.
            et (int):           EHMI type (0: text, 1: light).
            es (list):          EHMI toggle at any frame (0: off, 1: on).
            strn (int):         EHMI emission strength.
        """
        self.data_object = data_object
        self.a = a
        self.v = v
        self.s = s
        self.c = c
        self.e = e
        self.et = et
        self.es = es
        self.strn = strn

    def update_state(self):
        """
        Function to update the current state of the Car object.
        """
        self.s.append(self.s[-1] + self.v[-1] * dt + self.a[-1] * dt * dt / 2)
        self.v.append(self.v[-1] + self.a[-1] * dt)
        self.a.append(self.a[-1])
        self.es.append(self.es[-1])


# Create timer.
tic = time.perf_counter()
print('[calc_pos.py]: Initializing object trajectories.')

# Constants
t_end = 35                                      # running time: t [s]
r = 60                                          # frame rate; r [1/s]
n = r * t_end                                   # number of frames; n [-]
dt = 1 / r                                      # delta t; dt [s]

# Paths
output_dir = "F:\\render_output\\"

# Presets
preset = 1
ehmi_type = 0                                   # EHMI type (-1: off, 0: light, 1: text)

if preset == 1:
    delays_t = [15/6, 5/6, 0, 15/6]             # delays in seconds: [s]
    delays = [int(t * r) for t in delays_t]     # delays in number of frames: [-]
    routes = [2, 1, 0, 0]                       # route numbers
    stopping = [False, False, False, True]      # car stopping
    ehmi_trigger = [False, False, False, True]  # EHMI trigger when stopping (0: no, 1: yes)
    ehmi_types = [ehmi_type] * 4

elif preset == 2:
    delays_t = [35/6, 0/6, 30/6, 25/6]          # delays in seconds: [s]
    delays = [int(t * r) for t in delays_t]     # delays in number of frames: [-]
    routes = [2, 0, 1, 0]                       # route numbers
    stopping = [False, False, False, True]      # car stopping
    ehmi_trigger = [False, False, False, True]  # EHMI trigger when stopping (0: no, 1: yes)
    ehmi_types = [ehmi_type] * 4

elif preset == 3:
    delays_t = [0, 15/6, 45/6, 65/6]            # delays in seconds: [s]
    delays = [int(t * r) for t in delays_t]     # delays in number of frames: [-]
    routes = [0, 0, 1, 2]                       # route numbers
    stopping = [False, False, True, False]      # car stopping
    ehmi_trigger = [False, False, True, False]  # EHMI trigger when stopping (0: no, 1: yes)
    ehmi_types = [ehmi_type] * 4

elif preset == 4:
    delays_t = [5/6, 0, 30/6, 25/6]             # delays in seconds: [s]
    delays = [int(t * r) for t in delays_t]     # delays in number of frames: [-]
    routes = [1, 0, 1, 0]                       # route numbers
    stopping = [False, False, False, True]      # car stopping
    ehmi_trigger = [False, False, False, True]  # EHMI trigger when stopping (0: no, 1: yes)
    ehmi_types = [ehmi_type] * 4

elif preset == 5:
    delays_t = [5/6, 0, 30/6, 25/6]              # delays in seconds: [s]
    delays = [int(t * r) for t in delays_t]      # delays in number of frames: [-]
    routes = [1, 0, 1, 0]                        # route numbers
    stopping = [False, False, True, True]        # car stopping
    ehmi_trigger = [False, False, True, False]   # EHMI trigger when stopping (0: no, 1: yes)
    ehmi_types = [ehmi_type] * 4

elif preset == 6:
    delays_t = [15/6, 5/6, 0, 15/6]              # delays in seconds: [s]
    delays = [int(t * r) for t in delays_t]      # delays in number of frames: [-]
    routes = [2, 1, 0, 0]                        # route numbers
    stopping = [False, False, True, True]        # car stopping
    ehmi_trigger = [False, False, True, False]   # EHMI trigger when stopping (0: no, 1: yes)
    ehmi_types = [ehmi_type] * 4

elif preset == 7:
    delays_t = [15/6, 5/6, 0, 15/6]              # delays in seconds: [s]
    delays = [int(t * r) for t in delays_t]      # delays in number of frames: [-]
    routes = [2, 1, 0, 0]                        # route numbers
    stopping = [False, False, False, False]      # car stopping
    ehmi_trigger = [False, False, False, False] # EHMI trigger when stopping (0: no, 1: yes)
    ehmi_types = [ehmi_type] * 4

if ehmi_type == -1:
    ehmi_trigger = [False, False, False, False]

AN_route = bpy.data.node_groups["AN Tree"].nodes["Route Object List"]
for idx, val in enumerate(routes):
    AN_route.inputs[idx].object = bpy.data.objects["Route.00"+str(val)]

print('\n[calc_pos.py]: > Preset %s.' % preset)
# Initial and target conditions
l_routes = [114.62288, 91.11199, 106.71832]     # route lengths; l [m, m, m]

a_0 = 3                                          # acceleration; a [m/s^2]
a_d = -1 * a_0                                   # deceleration; a [m/s^2]
v_0 = 0                                          # velocity at t(0); v [m/s]
v_T = 8 + 1 / 3                                  # target velocity; v [m/s]; = 8+1/3 [m/s] (=
↪  30 [km/h])
s_0 = 0                                          # distance at t(0); s [m]
n_stop = 0
s_T = [l_routes[routes[0]], l_routes[routes[1]], l_routes[routes[2]], l_routes[routes[3]]] #
↪  target position; s [m]
for idx, val in enumerate(stopping):
    if val:
        s_T[idx] += -20 - 7 * n_stop
        n_stop += 1
s_b = -v_T * v_T  / (2 * a_d)                    # breaking distance; s [m]

e_max = 1e-10                                    # max error [-]

emission_strength = 10                           # EHMI emission strength

# Applicable when ehmi_type = 0 (light).
c_flash = 2                                      # EHMI flash count: c [-]
t_flash = 1 / 6                                  # EHMI single flash on duration: t [s]
n_flash = int(r * t_flash)                       # EHMI frames per flash: n [-]

```

```python
# Construct frame and time vectors.
frame_vector = range(1, n + 1)                       # list of frame numbers
time_vector = [frame * dt for frame in [0] + list(frame_vector)] # time vector

key_frames = [-1] * 4                                # key frame at phase switches; ([-]*)

# Loop over each object in Blender object list.
objects = []                                         # create empty objects list
objects_index = 0                                    # object iterator
check = [0] * 4

for car_object in (scene_object for scene_object_name, scene_object in
        bpy.context.scene.objects.items() if
         ↪  scene_object_name.startswith("VehicleContainer")):
    objects.append(Car(car_object,
                       [float("NaN")] * delays[objects_index] + [a_0],
                       [float("NaN")] * delays[objects_index] + [v_0],
                       [float("NaN")] * delays[objects_index] + [s_0],
                       stopping[objects_index],
                       ehmi_trigger[objects_index],
                       ehmi_types[objects_index],
                       [float("NaN")] * delays[objects_index] + [0],
                       emission_strength))

    print('\n[calc_pos.py]: >> Car: %.0f (%s)' % (objects_index,
     ↪  objects[objects_index].data_object))
    print('[calc_pos.py]: >> Route: %.0f' % routes[objects_index])
    print('[calc_pos.py]: >> frame_start: %.0f' % (delays[objects_index] + 1))
    print('[calc_pos.py]: >> a_0: %.0f m/s^2' % objects[objects_index].a[-1])
    print('[calc_pos.py]: >> v_0: %.0f m/s' % objects[objects_index].v[-1])
    print('[calc_pos.py]: >> s_0: %.0f m' % objects[objects_index].s[-1])
    print('[calc_pos.py]: >> stopping: %s' % objects[objects_index].c)
    if objects[objects_index].c:
        print('[calc_pos.py]: >> ehmi_type: %s (0: light, 1: text)' %
         ↪  objects[objects_index].et)

    # Loop over every frame, calculate and update Blender object state for that frame, log
     ↪  frame number on state change.
    for frame in frame_vector[delays[objects_index]:]:
        if objects[objects_index].c:
            if key_frames[0] < 0:
                key_frames[0] = frame

        # Calculate next state.
        objects[objects_index].update_state()

        # Acceleration phase.
        # When target velocity is reached set the following:
        # a = 0
        # v = v_T
        if objects[objects_index].v[-1] + e_max >= v_T \
                and objects[objects_index].a[-1] != 0:
            objects[objects_index].a[-1] = 0
            objects[objects_index].v[-1] = v_T
            if objects[objects_index].c:
                if key_frames[1] < 0:
                    key_frames[1] = frame + 1

        # Constant velocity phase.
        # When target distance is reached and object is meant to stop, set the following:
        # a = a_d
        if objects[objects_index].s[-1] + e_max >= s_T[objects_index] - s_b \
                and objects[objects_index].c \
                and check[objects_index] == 0:
            objects[objects_index].a[-1] = a_d
            if objects[objects_index].e:
                objects[objects_index].es[-1] = 1
            else:
                objects[objects_index].es[-1] = 0
            check[objects_index] += 1
            if key_frames[2] < 0:
                key_frames[2] = frame + 1

        # Deceleration phase.
        # When target velocity is reached, set the following:
        # a = 0
        # v = 0
        if objects[objects_index].v[-1] - e_max <= 0 \
                and check[objects_index] == 1:
            print('full stop frame %s' % (frame + 1))
            objects[objects_index].a[-1] = 0
            objects[objects_index].v[-1] = 0
            check[objects_index] += 1
            if key_frames[3] < 0:
                key_frames[3] = frame + 1

        # Reform eHMI strength based on eHMI type
        if frame == n:
            if objects[objects_index].c and objects[objects_index].e and not
            ↪   objects[objects_index].et:
                objects[objects_index].es[key_frames[2] - 1:key_frames[2] - 1 + 2 * n_flash
                ↪   * c_flash] = \
                    ([1] * n_flash + [0] * n_flash) * c_flash
                objects[objects_index].es[key_frames[2] - 1 + 2 * n_flash * c_flash:] = \
                    [0] * len(objects[objects_index].es[key_frames[2] - 1 + 2 * n_flash *
                    ↪   c_flash:])

    # Write calculated profiles and conditions to scene objects
    objects[objects_index].data_object["s_profile"] = objects[objects_index].s
    objects[objects_index].data_object["v_profile"] = objects[objects_index].v
    objects[objects_index].data_object["a_profile"] = objects[objects_index].a
    objects[objects_index].data_object["stop_trigger"] = objects[objects_index].c
    objects[objects_index].data_object["ehmi_trigger"] = objects[objects_index].e
    objects[objects_index].data_object["ehmi_type"] = objects[objects_index].et
    objects[objects_index].data_object["ehmi_toggle"] = objects[objects_index].es
    objects[objects_index].data_object["emission_strength"] = objects[objects_index].strn

    objects_index += 1

# Set scene frame rate and range to match the scripts.
bpy.context.scene.render.fps = r
if not key_frames[2] == -1:
    bpy.context.scene.frame_start = key_frames[2] - 1 - r * 5
else:
    bpy.context.scene.frame_start = r * 8 - 1
bpy.context.scene.frame_end = bpy.context.scene.frame_start + r * 10
bpy.context.scene.frame_current = bpy.context.scene.frame_start

# Set output filename
bpy.data.scenes["Scene"].render.filepath =
↪   f"{output_dir}preset_{str(preset)}_ehmi_{str(ehmi_type)}_frame_####"
```

```
if not key_frames[2] == -1:
    print('\n[calc_pos.py]: >> Stopping car metrics:')
    print('[calc_pos.py]: >> Acceleration at frame %.0f (t=%.3fs, s=%.3fm).'
        % (key_frames[0], time_vector[key_frames[0] - 1], objects[3].s[key_frames[0] - 1]))
    print('[calc_pos.py]: >> Constant velocity at frame %.0f (t=%.3fs, s=%.3fm).'
        % (key_frames[1], time_vector[key_frames[1] - 1], objects[3].s[key_frames[1] - 1]))
    print('[calc_pos.py]: >> Deceleration at frame %.0f (t=%.3fs, s=%.3fm).'
        % (key_frames[2], time_vector[key_frames[2] - 1], objects[3].s[key_frames[2] - 1]))
    print('[calc_pos.py]: >> Full stop at frame %.0f (t=%.3fs, s=%.3fm).'
        % (key_frames[3], time_vector[key_frames[3] - 1], objects[3].s[key_frames[3] - 1]))
else:
    print('[calc_pos.py]: >> No cars are stopping.')

print('\n[calc_pos.py]: >> Clip duration %.0f frames (%.3fs @ %.0ffps).'
    % (bpy.context.scene.frame_end - bpy.context.scene.frame_start + 1,
    time_vector[bpy.context.scene.frame_end] - time_vector[bpy.context.scene.frame_start],
    bpy.context.scene.render.fps))
toc = time.perf_counter()
print('\n[calc_pos.py]: Initialization complete: execution took %.3fs.' % (toc - tic))
```

Listing 36: Blender python script

C.3. BATCH

```
@echo off
setlocal enabledelayedexpansion

for %%f in (src/*.mkv) do (
        set fni=%%~nf
        if not '!fni!'=='background' (
                (echo file 'src/background.mkv' & echo file 'src/!fni!.mkv' )>list.txt
                ffmpeg -safe 0 -f concat -i list.txt -c copy !fni!.mkv
        )
)
del list.txt

pause
```

Listing 37: Batch script used to pad one second of predefined single-color frames to the start of each stimulus.

```
@echo off
setlocal enabledelayedexpansion

:: Set variables used throughout the script.
set gn=6
set gs=10

:: Loop over every .mkv file in the folder and apply blur filter
for %%f in (src/*.mkv) do (
        set fni=%%~nf
        ffmpeg -i src/!fni!.mkv -vf gblur=sigma=%gs%:steps=%gn% -advanced_editlist 0 -c:v
         ↪  libx264 -preset slower -crf 17 -x264-params
         ↪  bframes=0:keyint=30:min-keyint=3:rc-lookahead=30 !fni:clear=blur!.mkv
)

pause
```

Listing 38: Batch script used to create blurred version of original stimuli.

```
@echo off
setlocal enabledelayedexpansion

for %%f in (padded/*.mkv) do (
        set fni=%%~nf
        ffmpeg -i padded/!fni!.mkv -vf scale=-1:720 -preset slower -profile:v high -level:v
         ↪  5.0 -crf 18 -x264-params bframes=0:keyint=30:min-keyint=3:rc-lookahead=30
         ↪  !fni:1080p=720p!.mp4
)

pause
```

Listing 39: Batch script used to re-encode the stimuli to a lower quality version, to be used by SR Research Experiment Builder.