Hochschule Fulda
Fachbereich Angewandte Informatik

University of Massachusetts Boston
Department of Computer Science
Visual Attention Laboratory

Master of Science Thesis

Master of Science of Electronic Business

An algorithm for pattern recognition driven by eye movement

by

Andreas Lennartz

Supervisors:
Prof. Dr. Jan-Torsten Milde
Prof. Dr. Marc Pomplun

Boston, September 2007

Abstract

A lot of different works were published which examine basic search task where a user has to find a given object inside some picture or other object. During this task the subject's eye movement are recorded and later on analyzed for a better understanding of a human's brain and the corresponding eye movements to a given task. In such search tasks like "find-the-object" the question arises if it is possible to determine what a subject is looking for just by considering the given eye movement data, without knowing what he/she is looking for.

In this work an eye tracking experiment was introduced and conducted. The experiment presented different random-dot pictures to the subjects, consisting of squares in different colors. In these pictures the task was to find a pattern with a size of 3x3 squares. For the first part of the experiment, the used squares were in black and white, in the second part gray was added as an additional color. During each experiment the eye movements were recorded.

Special software was developed and introduced to convert and analyze the recorded eye movement data, to apply an algorithm and generate reports that summarize the results of the analyzed data and the applied algorithm.

A discussion of these reports shows that the developed algorithm works well for 2 colors and different square sizes used for search pictures and target patterns. For 3 colors it is shown that the patterns which the subjects are searching for are too complex for a holistic search in the pictures - the algorithm gives poor results. Evidences are given to explain this results.

Contents

Acknowledgements

Thanks to my parents for the mental and financial support.

Thanks to my supervising professors Dr. Marc Pomplun and Dr. Jan-Torsten Milde for their scientific support.

Chapter 1

Introduction

1.1 Motivation

A lot of research is done about the human recognition of patterns or snippets in pictures. For this purpose the movement of the gaze is tracked with special hardware (namely eye tracker), while the subject tries to find a snippet of a given picture. The subject searches for this snippet in the picture until he/she recognizes it, meanwhile eye movements are recorded. Analyzing the eye movement of these records afterward, it is obvious that only special parts of the picture are interesting for the subject, other parts are not. There is research done that tries to determine reasons for this, e.g. correlations between the color of the snippet and similar colors at the observed parts. The question arises if it is possible to determine the snippet that the subject was searching for, without knowing the snippet itself but the parts the subject has looked at.

The goal of this thesis is to determine a possible algorithm that takes eye movement input data of observed pictures in which the subject were to find a pattern. The eye movement data is analyzed and one or more possible patterns that could be the search pattern are given. In order to do this an experiment will be constructed where different random-dot pictures are given and the eye movements of the subjects looking for the search pattern in the picture are recorded. Then one algorithm will be developed, where existing research is considered. This algorithm will be tested for its functionality. Because of the great amount of possibilities the random-dot pictures (and their patterns) should be of a kind which can be easily processed by computers (e.g. black and white squares). Such an algorithm for detailed estimation of a user's intent would be valuable for the development of more efficient human-computer interfaces.

1.2 Chapter Overview

Chapter 2 introduces different aspects of visual search and eye tracking. **Section 2.1** gives a brief explanation of the human vision and how human beings conduct visual search, as well as the importance of visual search in scientific studies. **Section 2.2** explains how eye movements can be tracked and recorded using an eye tracking device. Existing research in the field of visual search is discussed in **section 2.3**. In order to create an algorithm that determines a target pattern out of eye movements, and based on the problems with current research, a new experiment is created for this work and described in **section 2.4**.

Chapter 3 explains the software developed and how it is used to build, run and analyze the newly designed eye tracking experiment. For this purpose three software parts were developed: An Experiment Builder to build experiments, described in **section 3.1**, an Experiment Engine to run an experiment, described in **section 3.2**, and an Experiment Analysis Tool to analyze the test data and create human-readable reports, described in **section 3.3**.

Chapter 4 explains the results of the experiment itself and how the developed algorithm works in different tests. It contains the experiment setup in **section 4.1**, the experiment runs in **section 4.2** and the experiment results in **section 4.3**.

Chapter 5 is the conclusion that summarizes and discusses the results of this work.

Chapter 2

Visual Search and Eye Tracking

2.1 Human Vision and Visual Search

2.1.1 Human Vision

The human eye is a very complex system. There is a region called *fovea* in the center of the retina which has a high density of photo receptors and therefore a high resolution. Outside this area (with a radius of about one degree of visual angle) the density of photo receptors decreases quickly, so the resolution is lower. This means that we have a very detailed vision in the center of our visual field and only coarse perception in the outer regions of the eye. For this reason the human eye is in a constant movement to receive a good vision of the environment.

Eye movements are not performed in a smooth manner like it perhaps can be assumed. The gaze position ”jumps” between inspected locations. These quick jumps are called saccades and the motionless phases in between are called fixations. Visual information can only be perceived during fixations, according to the brain's ability of constructing a stable image of the surroundings.[24]

A fixation duration can be considered as a measure of the effort to process information. The longer our attention rests at a certain location, the longer it presumably takes to deal with the presented visual information. (*Eye-mind hypothesis*, [7])

The saccade length reveals how thoroughly a certain region of the visual field is scanned. Short saccades indicate that the fovea is directed to positions close to each other, which means a high resolution scanning process. Long saccades imply that the local scene is only roughly perceived or that its information content is low. Fixation duration and saccade length are the basic eye-movement variables. Additionally, eye movements yield data about where and in which

temporal order a subject acquires visual information, e.g. eye movements reveal a subject's distribution and dynamics of visual attention.[12, p.5]

2.1.2 Visual Search

The visual search is a common and in the everyday live very important task for the human being. The success of localization of objects in a given area, e.g. the never-ending search for your keys, is determined by the active nature of looking. Because the human vision is a "foveated" visual system, it must be able to combine several eye movements to scan a visual field.

Most information during a visual search task are gathered during the fixations, and only little during the saccades, due to saccadic suppression and motion blurring. During each fixations in a search task, the scene is scanned for the target with the center of the eye for a high-resolution image, and the low-resolution periphery is used to plan the next subsequent fixation. An understanding of how the human visual system selects images could be important for a better understanding of the biological vision as well as a fundament for any artificial vision or human-computer interaction. ([2])

2.2 Tracking Eye Movements

In order to examine the eye movements of a human being the field of tracking eye movements (*namely eye tracking*) was introduced.

2.2.1 Definition Eye Tracking

As a very common entry point to gain quick knowledge of specific topics, the online editable-by-all dictionary Wikipedia is always a good starting point (though the information should always be double checked for its validity):

"Eye tracking is the process of measuring either the point of gaze (where we are looking) or the motion of an eye relative to the head. There are a number of methods for measuring eye movements. An instrument that does eye tracking is called an eye tracker." [21]

Eye tracking is part of the cognitive sciences, which resulted from the reasonable view that in order to understand the human brain scientists of different faculties like psychology, medicine and computer science should work together in interdisciplinary teams. These teams try to combine their knowledge and abilities to explore the brain's mechanism. [12, p.1]

In order to track and record the eye movements, special hardware is needed. These hardware is commonly called **EyeTracker**, and such a device used for this work is introduced in the next section.

Figure 2.1: EyeLink II eye tracking device

2.2.2 Eye Link II from SR Research

The Laboratory at University of Massachusetts Bostons uses such an Eye Tracking Device, namely EyeLink II from SR Research.

The EyeLink II system consists of three miniature cameras mounted on a padded headband. Two eye cameras allow binocular eye tracking or monocular by selecting of the subject's dominant eye. These cameras have built-in illuminators to even lighting for the entire field of view. Also combined with digital compensation for changes in ambient lighting, this results in stable pupil acquisition. (*Figure 2.1*)

The EyeLink Tracker from SR Research has a high resolution with a low noise level and a fast data rate (up to 500 samples per second, which means 500Hz) with the mounted video-based eye tracker. This data quality results in very low velocity noise, making the EyeLink II useful for saccade analysis and smooth pursuit studies. In addition, on-line gaze position data is available with delays as low as 3 milliseconds, making the system ideal for gaze-contingent display applications. [16]

The third camera, an optical head-tracking camera integrated into the headband, allows accurate tracking of the subject's point of gaze. The low noise level of this optical tracker means that the noise level of computed gaze position data is comparable to that of the original eye position data, allowing gaze data to be used for saccadic analysis. In contrast, the magnetic head trackers used in other head-mounted eye tracking systems have high angular noise and therefore limit the usability of gaze position data.

EyeLink II has a high compatibility with a wide range of subjects. Dark

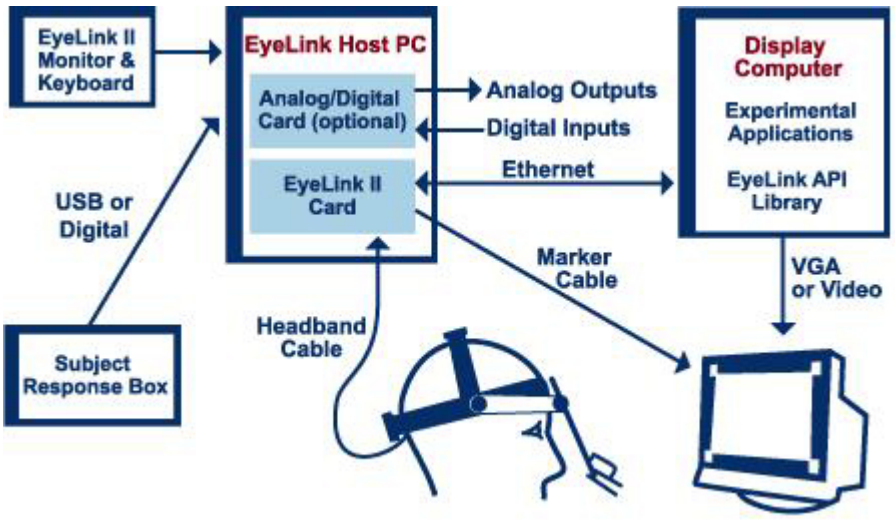


Figure 2.2: EyeLink II connection schema

pupil tracking and off-axis illumination allow tracking of subjects with almost all eyeglasses, eliminating the bright reflections. Eye camera sensitivity is high enough that even eyeglasses with heavy anti-reflection coatings that block up to 80% of infrared illumination are not a problem.

The EyeLink II eye tracking device (in the following called **EyeTracker**) is connected with its EyeLink II Software running on its host PC. The host PC can transfer data to the display PC, usually the computer where there experimental stimuli is displayed. Also a control device such as a game pad can be connected to the host (and/or display) PC. Both display and host PC are connected with a standard Ethernet cable. The monitor screen of the display PC has a set of markers attached to it, which are delivering additional information of the subjects' head position to the EyeTracker. (See *figure 2.2*).

Figure 2.3 shows an example for the camera setup screen of the EyeLink II software.

2.3 Visual Search Experiments

In order to examine the visual search by humans, experiments using eye tracking systems such as the EyeTracker were created. In these experiments which examined the detection of targets and involved distracters, first of all it was found out that the saccadic movement during a visual search task can be influenced and guided by visual cues ([23]). These cues can be of different types. The most important characteristics of these cues are color ([22]) and shape ([5]). These visual cues can affect the search performance ([20]). However, the experiments do not reveal what image feature or structural cues are attracting the observer's

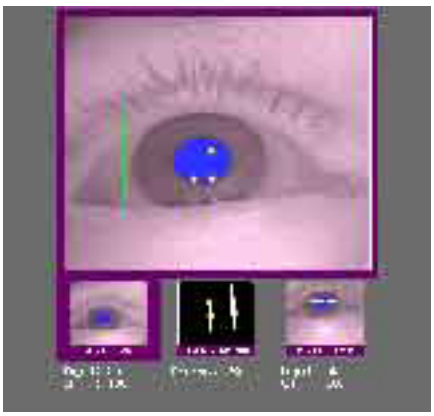


Figure 2.3: EyeLink II camera setup screen shot

gaze.

Eye movements are crucial for any (foveated) visual system as a salient feature of visual search. There has been considerable investigations in free viewing studies to examine what kind of image features influence human eye movements. Free viewing studies analyze the eye movements of a human subject without a given task. An analysis at the point of gaze shows that eye movements are not really random - they are possibly drawn to image patches with a high contrast and low interpixel correlation. ([11],[15])

The question is how such free viewing experiments can be correlated to visual search problems - if a search task is given in first place, the selectivity of the eye movements is strongly influenced. Therefore, free viewing studies are not ideal to examine visual search. In studies with a given task, it could be examined that eye movements and especially the fixations of the eye try to analyze parts of the stimuli which are similar to the target. This means that we peripherally select areas which are similar looking (or are the same) to the search target, and then our fixation will analyze if the selected area really correlates to the target. [13]

Based on these perceptions, research has been done to prove that eye movement is depending on the search task and the stimuli. In an experiment by Rajashekar, Bovik & Cormack (*see [14]*) the eye movement of three observers were recorded while they tried to find a simple geometric shape (circle, dipole, triangle, or bow tie) in a "noise stimuli" picture. The shapes are shown in *figure 2.4*. The stimuli were based on a Fourir spectrum where the noise matched the average spectrum of natural images. An example stimuli with observer's eye movements superimposed is seen in *figure 2.5*. A quick examination of the stimuli might show the difficulty of the search task. Further it could be assumed that in such a low signal-to-noise ratio stimuli observers might be simply deploy their eye movement uniformly across the stimuli. But the analysis showed

Figure 2.4: Simple geometric target shapes, [14]

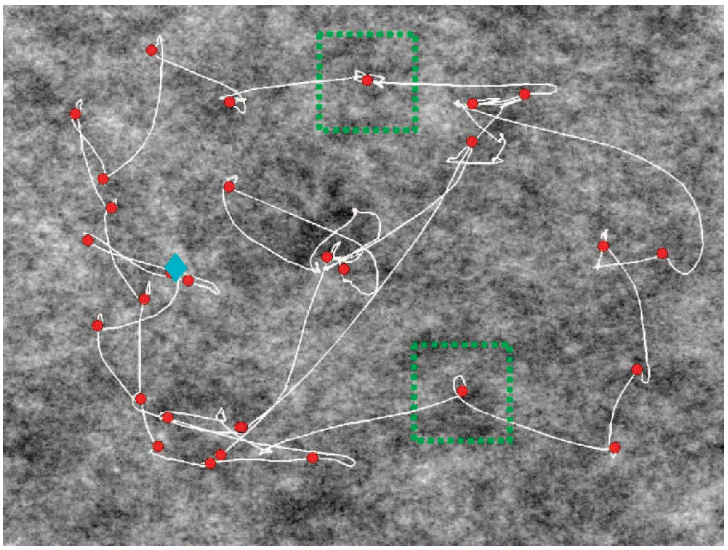


Figure 2.5: Noise stimuli, with superimposed eye movements from a trial, [14]

that a different strategy is applied.

For the analyzation of the experiment, a classification image technique was used by incorporating eye movements as described in the following. For each subject and target, a region with the size 128x128 pixel (twice the size of a target) around each fixation was used. These regions were averaged across the trials to yield classification images. All fixations by the subjects, including the final fixation and those that landed on the target, were used in the analysis. In order to enhance the details in the classification images, a statistical thresholding was performed by setting pixel intensities within one standard deviation of the mean equal to the mean (*referring to [1]*).

The statistically thresholded results for all subjects (namely LKC, EM and UR) and targets (circle, dipole, triangle, and bow tie) are shown in *figure 2.6*. The first row illustrates the four targets that observers were instructed to find in the stimuli. Each of the other rows shows the classification images for the three observers for these targets.

In order to quantify the uncertainty associated with the shapes of these classification images, the authors bootstrapped* the averaging procedure and com-

*Bootstrapping is the practice of estimating properties of an estimator (such as its variance) by measuring those properties when sampling from an approximating distribution. One standard

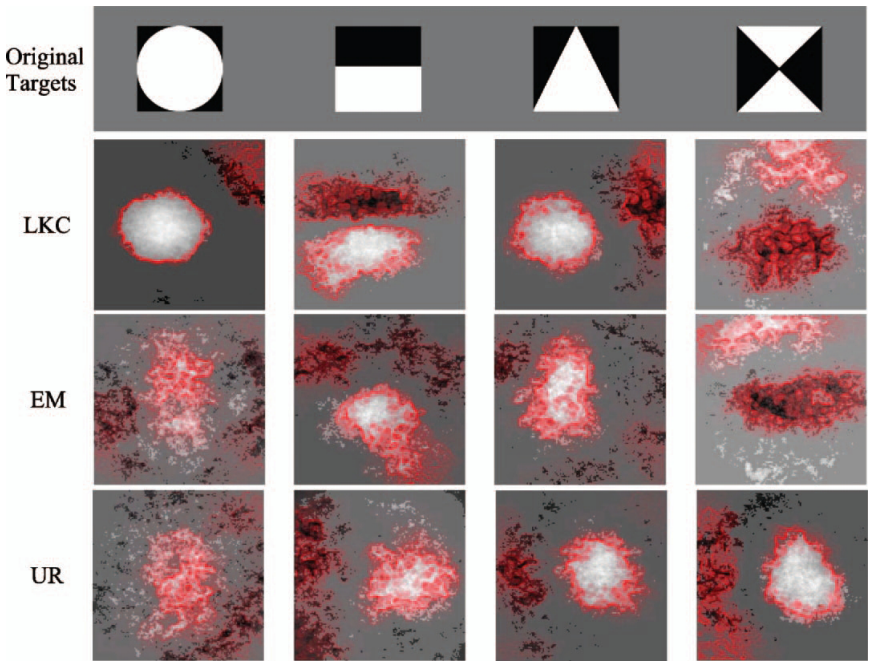


Figure 2.6: Statistically thresholded results for all subjects, [14]

puted the boundaries of each of the resulting thresholded classification images as follows. To detect the boundaries of the statistically filtered classification images, the two largest regions in the statistically filtered image were detected using a connected components algorithm. The outlines of each of these regions were then used to represent the boundary of the classification image. Bootstrapping was then used to verify the robustness of these boundaries. The boundaries of the resulting classification images were then added together and superimposed on the classification images to reflect the stability of the boundary. The aggregate of all of the bootstrapped boundaries is shown superimposed in red in *figure 2.6*.

The red well-defined boundary indicates that the structure in the classification image is robust, and diffused edges indicate that the shape of the classification image in that region varies across trials. Rajashekar, Bovik, & Cormack argue that the results show that subjects fixated areas which look like the target shape. The results should prove that there is a computable connection between the fixations and the target object, and that subject's fixations are tending to look at areas in the stimuli which are highly similar to the target shape. But *figure 2.6* on which these assumptions are based shows that a qualitatively proof is missing.

Summarizing, research shows that there is a connection between the given

choice for an approximating distribution is the empirical distribution of the observed data. [4]

search task and the subject's eye movement. Regarding Rajashekar, Bovik, & Cormack ([14]), and also Pomplun ([13]), it is likely that a subject's gaze tends to fixate shapes in a stimuli that are similar to the target shape. But how much information does the correlation between eye movement and target shape really contain? Is it possible to find a measurable way to qualitatively examine this information? The question arises if it possible to determine the target shape by analyzing only the subject's eye movement, where a priori the target shape is unknown. This would give valuable statements about the connection between the target and the stimuli. Based on this, the following experiment for an Eye-Tracker device was created.

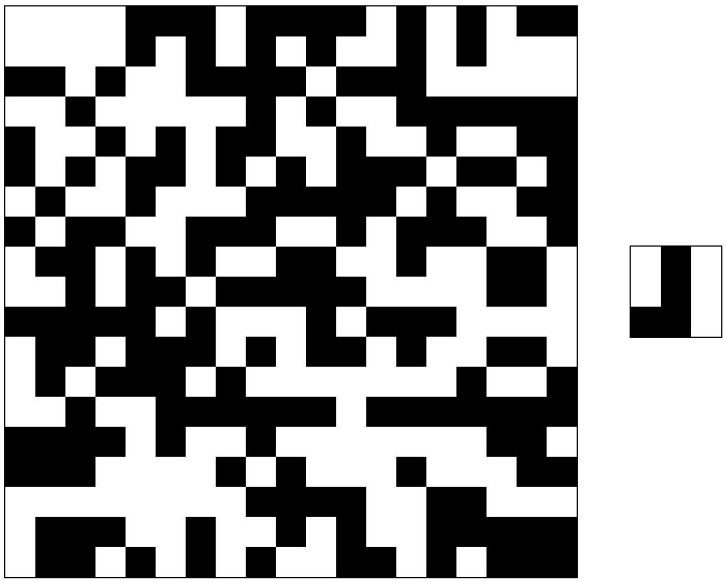


Figure 2.7: Example of target pattern and search picture with 2 colors

2.4 Experiment Description

For this work an eye tracker experiment is supposed to run with several subjects. The subjects' task is to search a random-dot pattern consisting of 3x3 squares in a stimuli picture. The number of colors are either 2 or 3 (white, black and gray as additional color). *Figure 2.7* shows an example of the target pattern and stimuli picture for 2 colors, *figure 2.8* for three colors. The search task will be repeated 30 times for each subject, each time with the same target pattern but a different stimuli.

The random-dot pattern is used in order to make quantitative statements about the selectivity of the eye movements and the information they are containing. For this reason the stimuli changes for each search task (2 or 3 colors) but the search pattern is kept. Furthermore, by repeating the search task with the same search pattern it can be memorized easily by the subjects. The number of colors variates between 2 and 3 in order to analyze how the changed complexity influences the search behavior and the results.

During the whole experiment the eye movement of a subject is recorded. Later, this data has to be analyzed, and different algorithms are applied. Reports summarizing the data are generated.

These reports will contain:

- **Fixation data**
  A list of all fixations in the stimuli picture and the surrounding area.

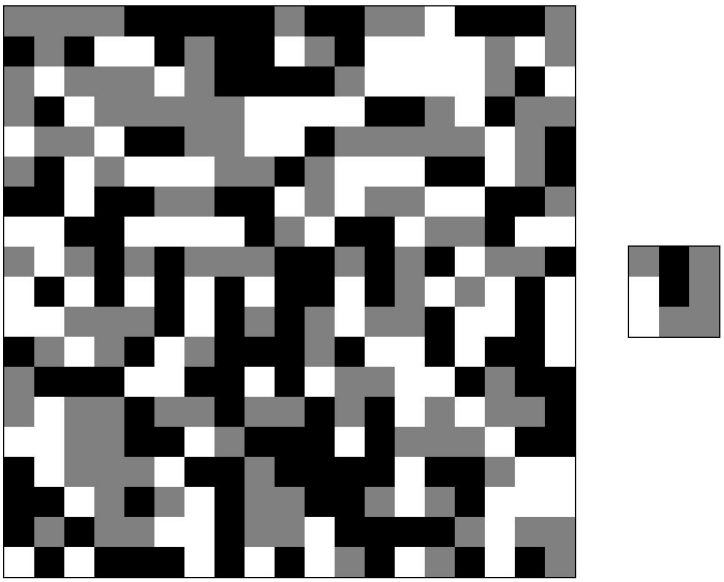


Figure 2.8: Example of target pattern and search picture with 3 colors

- **Fixation rankings**
  A list of the fixations and the surrounding area ordered by either their frequency or the fixation duration.

- **High similarity ranking**
  For each fixation and it's surrounding area, a list with patterns that are similar to the target pattern can be generated. Similar means e.g. if a fixation with a 3x3 area in the stimuli picture is given, a similar pattern would be one where 8 out of 9 squares for each position are the same. A list with all similar patterns will be created, and the reports present the most frequent patterns inside this list. This list will be called *high similarity list*, and the corresponding ranking *high similarity ranking*. Also, instead of observing the frequency of the patterns for building the ranking, the duration of each fixation can be inherited to each similar pattern - then a ranking where the similar patterns are ordered by their inherited duration can be built.

- **Sub-fixation rankings**
  For each fixation and its surrounding area, a list of several sub-fixations can be extracted. These sub-fixations are one square smaller than the target pattern itself, in a case of a 3x3 target pattern the sub-fixations size is 2x2 squares. A list of the most frequent sub-fixations can be generated. Also, for these sub-fixations the position inside the fixation area can be used to build rankings separately for each position.

The next chapter discusses in detail the software and algorithm developed for this work to build, run and analyze this experiment.

Chapter 3

Experiment Software

For this work three different stand-alone applications were developed. These are:

- **Experiment Builder**
- **Experiment Engine**
- **Experiment Analysis Tool**

The Experiment Builder is an application to create experiment files which can be run by the Experiment Engine. The Experiment Engine displays the experiment and communicates with the EyeTracker to gather the eye movement data. The Experiment Analysis Tool creates output files that apply different analysis algorithms to the experiment data and displays the corresponding results.

In the following each of these applications and their intercommunication is discussed in detail.

3.1 Experiment Builder

The Experiment Builder provides the functionality to create different trials for the experiment within a convenient GUI. With this application, the parameters for each trial of an experiment is set up and the necessary data files to display the experiment are created.

3.1.1 Step by Step Experiment Builder

The program is started by running `BMPBuilder.exe`. It is located on the enclosed CD in the Folder "BMPBuilder". To run properly, it requires Windows

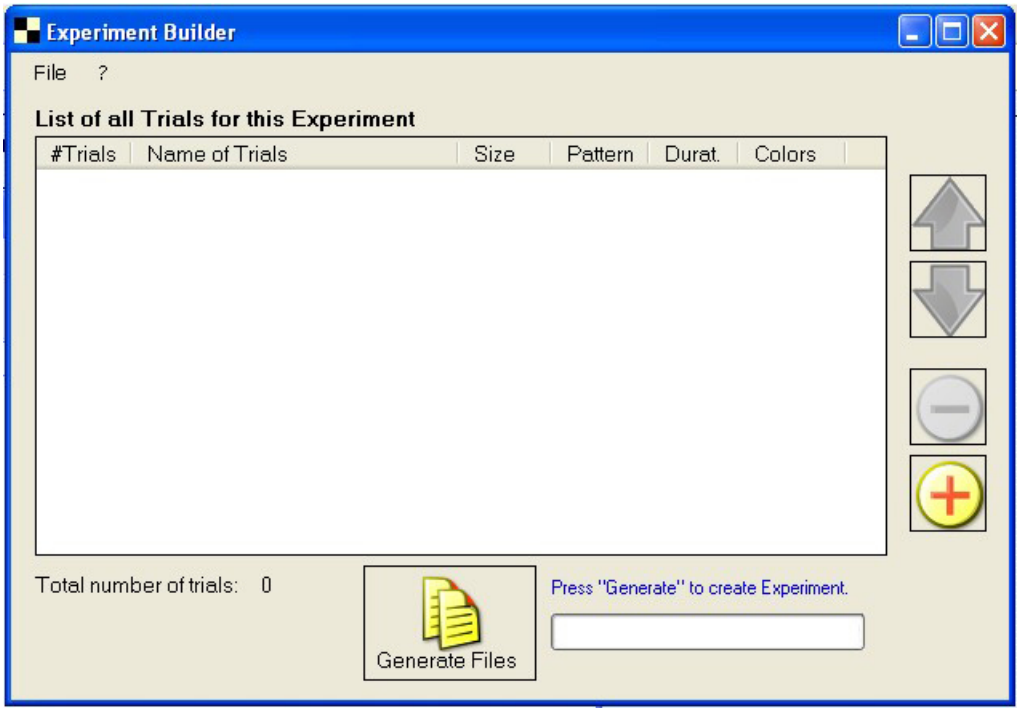


Figure 3.1: Startup screen of Experiment Builder

XP and the .NET 2.0 Framework.

In the main GUI that shows up after start of the application, the user begins with an empty experiment. (*Figure 3.1*)

Main Window

In the main window, the user has the following choices:

- Adding one ore more trials (a so called **block of trials**) to the experiment. A block of trials contains trials with the same set of parameter. When adding a block of trials, a new window will open and ask the user for the specific parameters. After entering and confirming these parameters, the trial block will appear in the main GUI under *"List of trial blocks"*.
- Moving up or down a block of trials will change the sequence in which the experiment is run.
- Deleting of an previously added block of trials.
- Creating of necessary data files for the Experiment Engine.
- Via pull-down menu "File" an existing experiment can be loaded, a previously created one saved or a new one started.

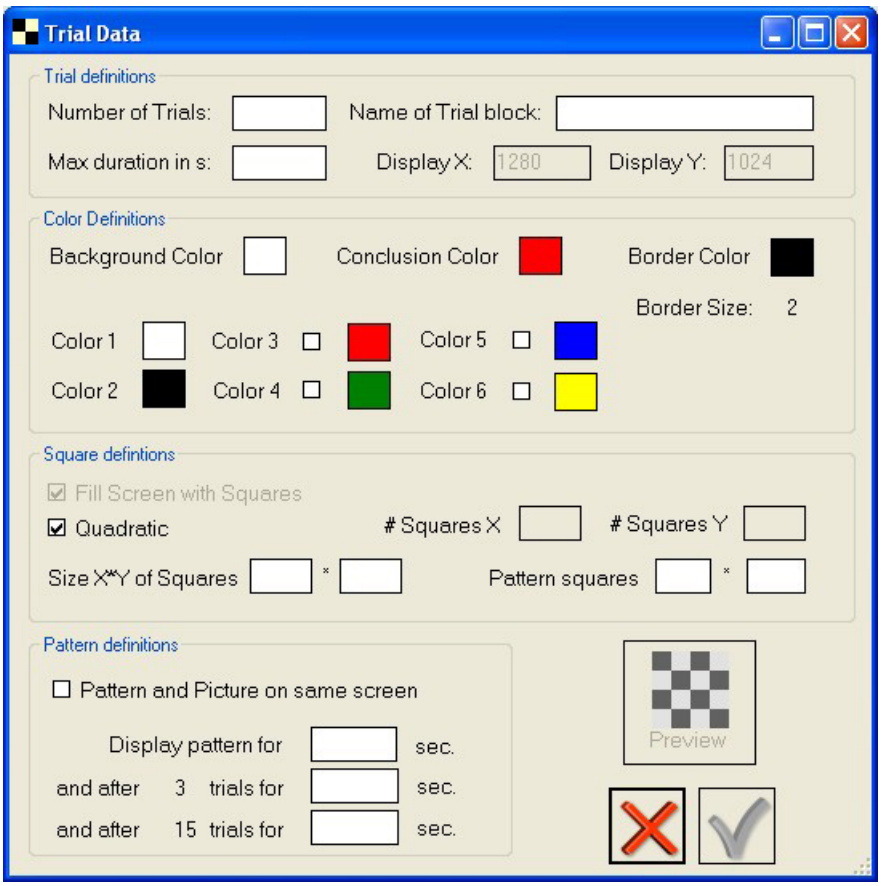


Figure 3.2: Window to enter trial block parameter

.

Adding a Block of Trials

By clicking on the *"Plus"* symbol in the right half of the main window (this button is always active), a new window will open and prompt the user to enter different parameters (*figure 3.2*). The opened window is a modal dialog and will not let the user return to the main window until either it is closed (which will not add any trials to the experiment) or until all necessary data fields are filled.

In order to create a new block of trials, the following information must be provided by the user (*figure 3.3*):

- *Number of trials*
  Number of trials to which parameters are applied.

- *Name of trial block*
  A name to identify this block of trials.

- *Max duration in s*
  The maximal duration of each trial in seconds before it is timed out - the ”solution” of the trial is displayed and the next trial starts.

- *Display X/Y* **(Text fields disabled)**
  Defines the resolution of the screen for displaying the experiment. This value is fixed with 1280x1024 pixel.

- *Color definitions*
  Each color for the squares in the random-dot pictures and patterns can be defined. The first two colors are always activated. By checking one or more of the check boxes additional colors can be added. Clicking on a color itself will display a color dialog where the user can change the color for the specific selection. Not only the colors for the squares can be set, also the color for the ”solution” frame (a frame that shows where the pattern is located in the picture, displayed at the end of a trial), the color for the surrounding box of the picture (with the displayed border size) and the background color can be specified.

- *Square definitions*
  In the *”Size X*Y of Square”* text fields, the user can enter the width and height in pixel for each square. If the *”Fill Screen with Squares”* check box is activated (which is in the current program the only possible configuration), the whole screen is used and the total amount of squares for the picture is calculated and displayed in the *”# Squares X/Y”* text fields. (Therefore, the number of squares in these text fields cannot be modified.) If the *”Quadratic”* check box is activated, the picture’s overall shape is quadratic, otherwise it will have the rectangular shape of the screen. The number of squares of which the pattern itself consists is entered in the *”Pattern squares”* text fields. They define the number of squares in the pattern - e.g., the values *3 * 3* would define a pattern with each three squares in horizontal and vertical direction.

- *Pattern definitions*
  The user can specify if the patterns and pictures are either displayed on the same screen or on subsequent screens. If checked, the text fields below are disabled and will not need data input. If not, these text fields define the time periods the pattern will be displayed before it is replaced by the picture.

- *Preview and cancel/accept*
  The preview button generates a rough preview of the look-alike of one trial of the current block. The preview is optimized for the fixed display size 1280x1024. The cancel button will close the trial parameter window and

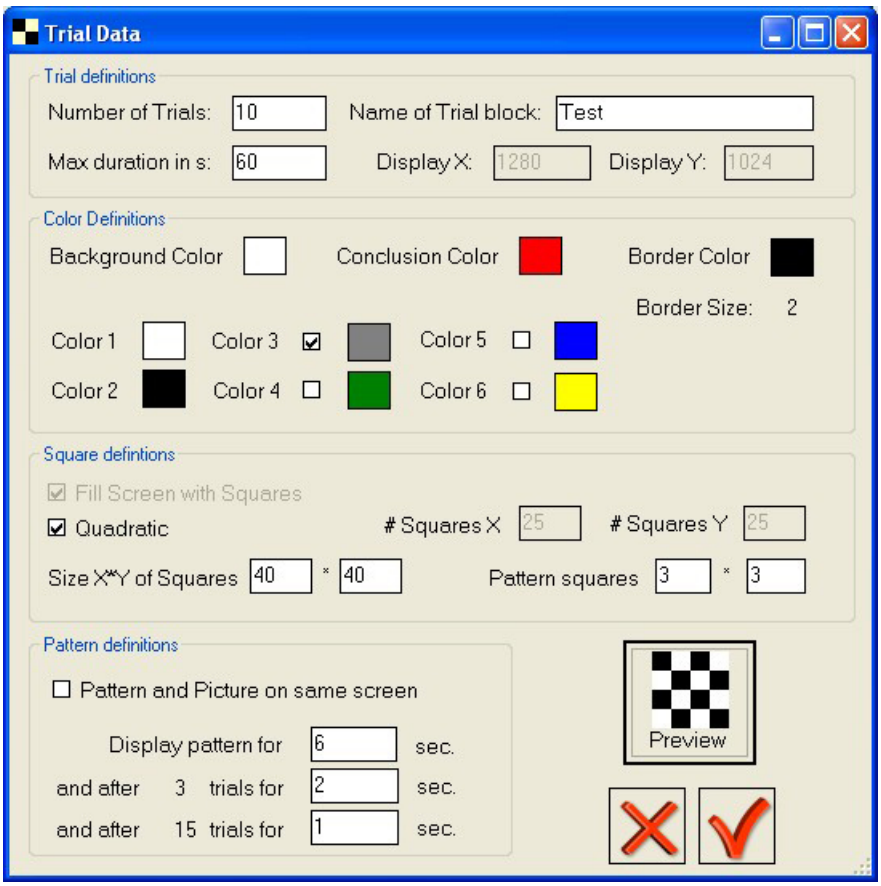


Figure 3.3: All necessary parameter information is entered

discard all entered information. If all necessary data is entered, the accept button is enabled and can be used to add the data to the experiment.

After accepting the data input, these parameters are used to create a block of trials that appears in the trial list of the main window. The total number of trials is stated below the list of all trials. (*"Total number of trials"*, *figure 3.4*)

Additional Data Modifications

Selecting one of the previously added trial blocks allows the user either to delete it or to move it up or down. The order in which the trials are executed during the experiment is the same as they are listed in the trial list. Moving up or down of a trial block will change the execution order. Deleting will remove it from the experiment. To edit existing experiment data, the user can double-click on the trial block that needs to be modified.

Via the pull-down menu *"File"* it is possible to start over, load or save an experiment (*figure 3.5*). The user can exit the program at any time. A warning will

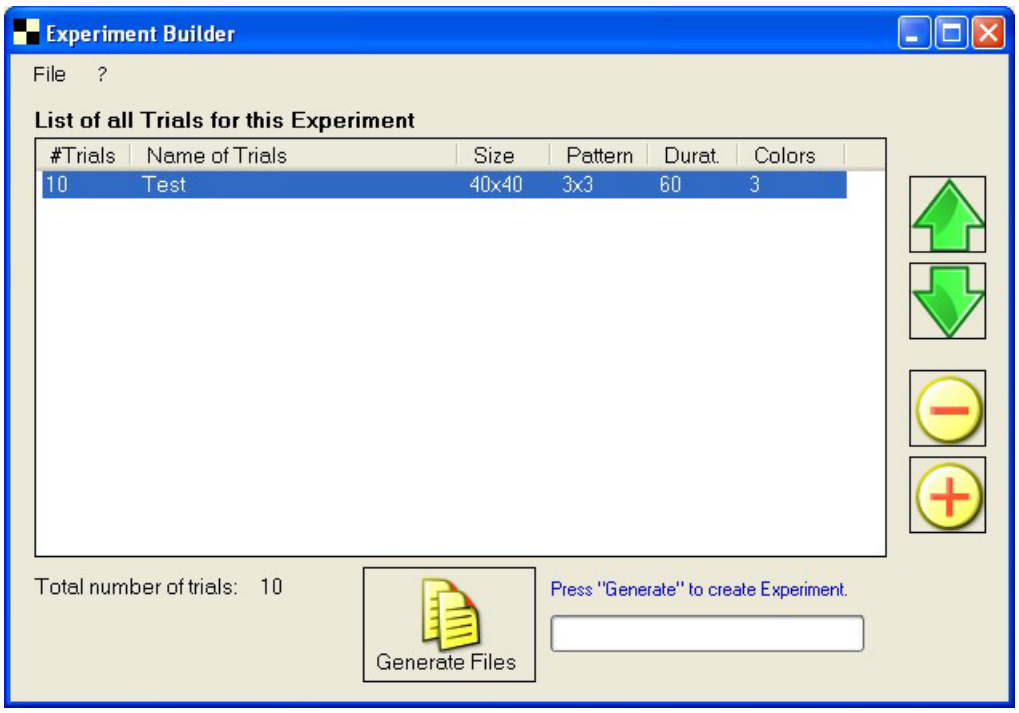


Figure 3.4: A new block of trials has been added to the trial list

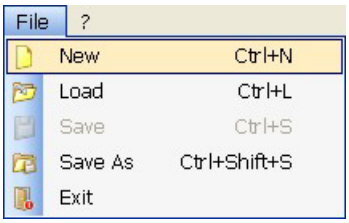


Figure 3.5: The File menu

appear if there are unsaved changes. The *"?"* menu accesses an *"Info"* window which displays contact and program information.

Generating Experiment Data

Finally, when all desired trial blocks are added to the experiment, the user can generate the experiment files. Clicking on the *"Generate"* button opens a folder browser dialog where the user can choose the folder to create the experiment files in.

3.1.2 Experiment Builder Files

The Experiment Builder will create the following directory structure and the following files:

- `/config.txt`
  Provides display size and total trial number data in a text file. Will be read by the Experiment Engine.
- `/experiment.bin`
  Contains the experiment object data which can be read by the Experiment Builder.
- `/data/PatternData_`**X**`.bin`
  Contains the pattern object data for trial number **X**. Is processed by the Experiment Analysis Tool.
- `/data/PictureData_`**X**`.bin`
  Contains the picture object data for trial number **X**. Is processed by the Experiment Analysis Tool.
- `/data/trialData_`**X**`.txt`
  Contains text data for each trial **X** that is needed by the Experiment Engine and contains e.g. the location of the search pattern in the displayed picture.
- `/images/trial_`**X**`.bmp`
  The search picture image for each trial **X** in a bitmap file. The Experiment Engine will display this search picture during the experiment execution.
- `/images/trialPattern_`**X**`.bmp`
  If the target pattern and the search picture are not displayed on the same the screen, this image file containing the search pattern will be displayed at the beginning of each trial **X**.

3.1.3 Class Overview

The following section gives a overview of the code structure and the classes of the Experiment Builder. Additional to the structural overview the complete code documentation can be found in *appendix D*. The complete source code, its project files and its documentation in electronic form are also on the enclosed CD. (*CD-Content is described in appendix A.2*)

The Experiment Builder is written in C#, using the .NET 2.0 Framework from Microsoft. It was developed with Visual Studio 2005. A list of all used software and external tools is listed in *appendix A.1*.

The Experiment Builder can be divided into three sub-areas: GUI objects, experiment data objects and pattern/picture objects.

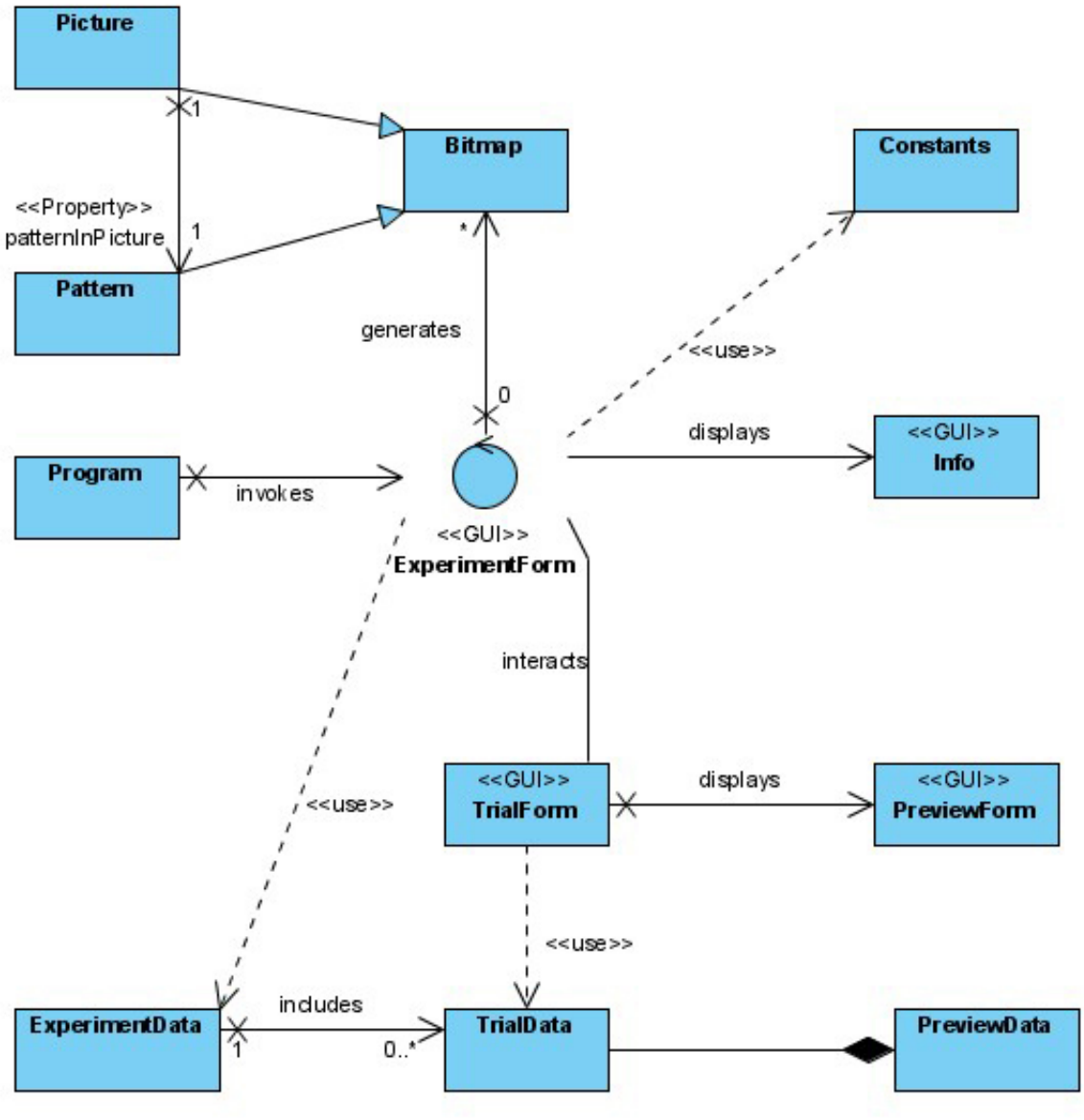


Figure 3.6: Overview of all classes in the Experiment Builder

The GUI objects represent the user interface, marked by the stereotype `<<GUI>>` in the class diagram (*figure 3.6*). Its functionality is described in the previous *section 3.1.1*. The `ExperimentForm` object displays the main window and the `TrialForm` object is responsible for the trial parameter window. For displaying a preview, a new `PreviewForm` object is created, and to display the info window the `Info` object is used.

`Constants` provides the GUI with already pre-defined data (such as the

display size of the experiment screen), mostly provided in static methods. The GUI itself is invoked by `Program`, which does nothing else than to instantiate a new `ExperimentForm`.

The `ExperimentForm` and `TrialForm` user interface objects are using the `ExperimentData` and `TrialData` to store the entered data. An `ExperimentForm` starts with an empty `ExperimentData` object, containing an empty list of `TrialData`. When the user adds a new block of trials, an instance of a `TrialForm` is displayed. After entering all information into the user inferface object `TrialForm`, the data is stored in a `TrialData` object and then added to the list of trial blocks in the `ExperimentData`. `TrialData` contains also a `PreviewData` object, where the `PreviewData` stores the trial data that is necessary to display a preview. Hence, the `PreviewData` is used to show a preview within the `PreviewForm`. The `ExperimentForm` displays the changed list of trials within its GUI.

The `Pattern` and `Picture` objects represent target patterns and search pictures that the Experiment Builder is supposed to generate. They provide functionality to create representative data objects and images of a pattern or picture, where its data and its corresponding bitmap files can be stored on the hard drive and later on read and accessed by the Experiment Analysis Tool.

A `Picture` or `Pattern` consists of a two-dimensional array of squares, each square can hold a color value. Because both have many common properties and methods, both inherit from the class `Bitmap`. After instantiation of a `Picture` or `Pattern` (using the values the user entered in the GUI) they can be filled with evenly distributed random color values. After that, a pattern can be connected with a picture, so that the pattern itself occurs exactly once inside the picture. When both (picture and pattern) are created and connected, the generation process (invoked by the user in the GUI) uses these objects to create and store their bitmap image objects and their data files (`bmp` and `bin` files in the ”data” and ”image” folder).

3.2 Experiment Engine

The Experiment engine displays and runs an experiment created by the Experiment Builder. While running the experiment, it communicates with the EyeTracker device and takes care of controlling the EyeTracker according to the experiment needs.

3.2.1 Step by Step Experiment Engine

Before the Experiment Engine can start, the previously generated files and folders from the Experiment Builder have to be copied in the same folder where the executable of the Experiment Engine is located. After starting the Experiment

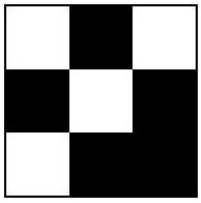


Figure 3.7: Example for a target pattern

Engine, it will access the file `config.txt` in the same folder as well as all `txt` files in the ”data” Folder and all `bmp` files in the ”image” folder.

To run the experiment, the executable `BMPBuilder.exe` must be started. To run properly, the screen resolution of the host computer should be set to 1280x1024. An existing connection to an EyeLink II eye tracking device from SR Research is needed to record eye movements, but there is also a simulation mode in which the Experiment Engine can be run. In the simulation mode the experiment execution will run without an EyeTracker, but there will obviously not be any data recording. The compiled program file `BMPBuilderSim.exe` enclosed on the CD is compiled in simulation mode. But because of the close dependency on the hardware and the installed EyeTracker libraries, it is very likely that the executable has to be recompiled to run.

After execution of the program a brief description of the experiment is given. Pressing ”Escape” will continue with the calibration screen, in which the camera of the EyeTracker can be focused. Also, the calibration of the gaze fixation to properly record data has to be adjusted. (The EyeTracker calibration is documented in [19].) Calibration is not available in simulation mode.

After the calibration each trial will start with the drift correction. The drift correction tries to correct occasional shift of the headset on the subject’s head. The subject will look at a dot in the center of the screen and continue to the next trial by clicking a button on a game pad connected with EyeTracker. (”Escape” will skip the drift correction in simulation mode.)

After drift correction the target pattern will be displayed to the subject. For an example of a target pattern see *figure 3.7*. The subject should memorize this pattern - the display time depends on the parameters defined in the Experiment Builder. After the given time period the search picture will appear in which the subject is supposed to find the target pattern. If pattern and picture are defined to be on the same screen, it will directly start the trial. See *figure 3.9* for an example screen.

As soon as the subject finds the target pattern in the picture, the trial can be ended by pressing the button on the game pad (or ”Escape” in simulation mode). An example for a search picture screen (in which the pattern in *figure 3.7* can be found) is shown in *figure 3.8*. To display the ”solution” of the trial, a red

Figure 3.8: The search picture in which the target pattern has to be found

rectangle will mark the position of the target pattern in the picture and therefore provide feedback to the subject whether he or she was right or wrong. If the subject does not find the pattern (and does not press the button) the experiment will timeout after the defined time period. The position of the target pattern will be marked and the experiment continues with the drift correction for the next trial.

At the end of the experiment a *"Save File"* dialog pops up, and the experimenter can choose a folder where the recorded eye data should be stored. The eye data comes in an `edf` binary format. A tool provided by SR Research called Visual2ASC can convert these `edf` files into ASCII text files (Ending `asc`). The `edf` file will contain samples and events of the eye movement, where a sample describes each position of the eye for each scan period (e.g. 250Hz). Also the EyeTracker can calculate fixations, blinks and saccades of the eye and names them as events. To analyze the recorded tracking data with the Experiment Analysis Tool, the `edf` file has to be transformed into an `asc` file, whereby only the events are needed.

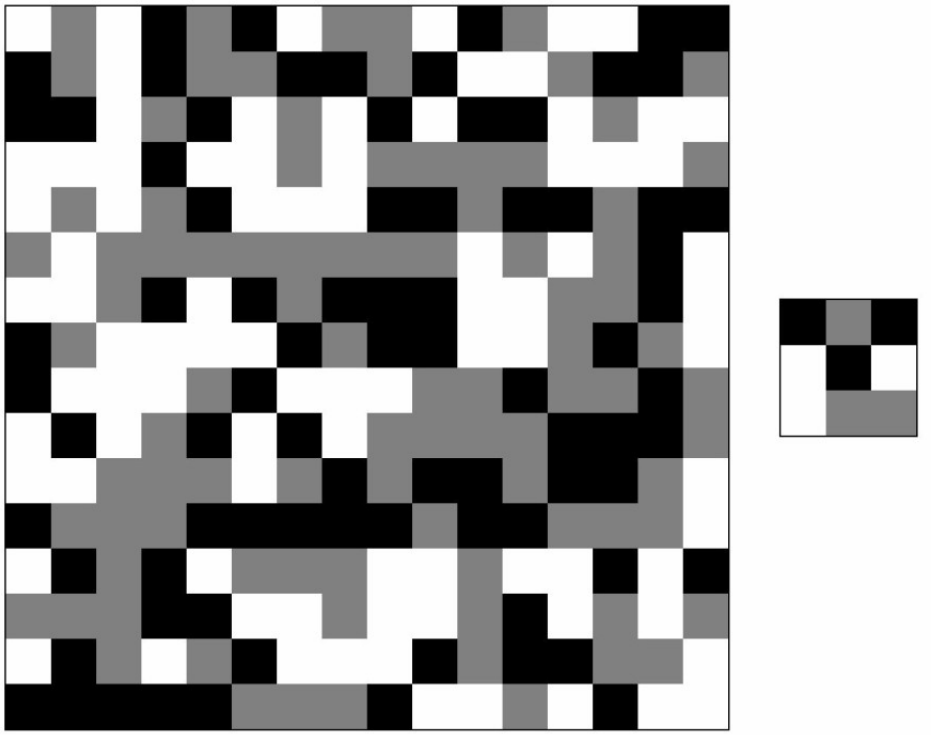


Figure 3.9: Target pattern and search picture on the same screen

3.2.2 Code Overview

The Experiment Engine is written in C. The source code and the Visual Studio project files as well as a pre-compiled executable can be found on the enclosed CD in the folder "BMPExperiment". The Experiment Engine uses libraries from SR Research for communication with the EyeTracker. More information, a programming guide and the EyeTracker software is available on the SR Research Homepage (*see [18]*). To display graphics on the screen, the Open-Source library SDL (Simple DirectMedia Layer, *see [3]*) is used. SDL requires an OpenGL-compatible graphic card and drivers.

To develop the Experiment Engine, an existing code was used and modified. This code template called "picture" is included in the EyeTracker software packet for developers. It was modified according to the guidelines in the EyeTracker Programming guide (*see [17]*). The template contains code for displaying pictures on the screen, and records the eye movement of the subject. (For complete documentation of this template *see [17, Chp. 17]*.)

In the following a brief description of the code files is given:

- *picture.h*
  Declarations to link together the template experiment files.

- *main.c*
  Main function for windows execution. Setup and shutdown of link to the EyeTracker device, display of experiment graphics, opens and saves `edf` file.

- *trials.c*
  Called to run all trials for an experiment. Performs system setup at the start of each experiment, then runs the trials. Handles return codes from trials to allow the subject to end the trial by pressing a particular button. The code that is changed compared to the template is inside this file.

- *trial.c*
  Implements a trial using the SDL graphics library. Includes the code for drawing and displaying the experiment picture, for sending messages to the eye tracker and handling any subject responses.

The experiment first initializes and connects to the EyeTracker. It then creates a full-screen window, and sends a series of commands to the tracker to configure its display resolution, eye movement parsing thresholds, and data types. Then it asks for a file name for the `edf` file, which it commands the EyeLink tracker to open on its hard disk.

The program then runs all the experiment trials. Each trial begins by performing a drift correction, recording is then started. After all trials are completed, the `edf` file is closed and transferred via the link from the EyeTracker hard disk to the display PC. At the end of the experiment, the window is erased and the connection to the EyeTracker is closed.

3.3 Experiment Analysis Tool

The Experiment Analysis Tool takes the data generated during the experiment, processes them using different algorithms and creates reports . Because there is no ”standard” way to develop reports showing the algorithm’s results, the Experiment Analysis Tool generates Excel files that make data interpretation easier for humans.

3.3.1 Step by Step Experiment Analysis Tool

The Experiment Analysis Tool is started by running `BMPAnalysis.exe`. It is located on the enclosed CD in the folder ”BMPAnalysis”. To run properly, it requires Windows XP and the .NET 2.0 Framework. For generating Excel sheets, Microsoft Office 2003 and the Microsoft Office Primary Interop Assemblies (PIA) for Office must be installed. (*See [10]*)

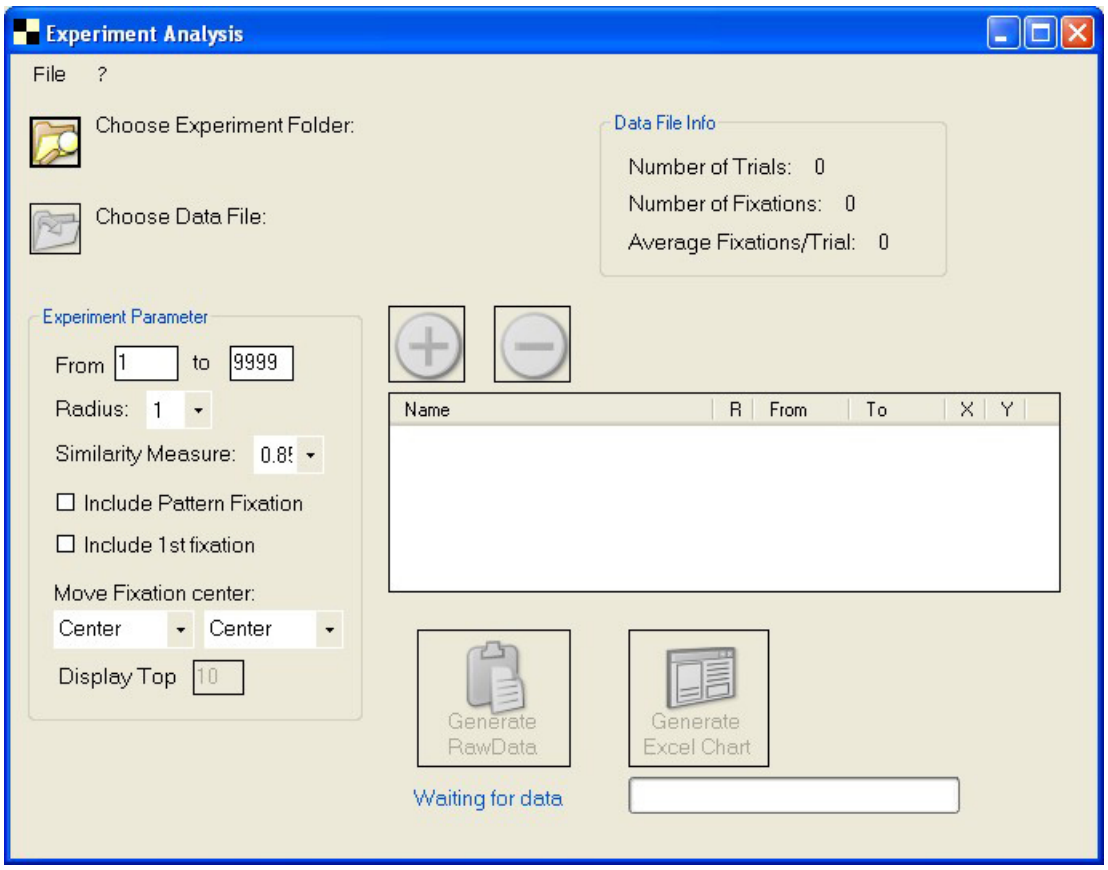


Figure 3.10: Experiment Analysis Tool main GUI

In the main GUI that shows up after start of the application, the user begins with no experiments in the analysis queue. (*Figure 3.10*)

The next step is to select a folder that contains experiment data. For this purpose the user can click on the folder symbol next to *"Choose Experiment Folder"*. A browse folder dialog will appear, where the desired experiment folder can be chosen. After selecting a folder, the second folder symbol next to *"Choose Data File"* is enabled - clicking on it let the user choose an `asc` data file containing the EyeTracker events from the experiment.

After folder and file are selected, the *"Data File Info"* box displays general information about the experiment, specifically the number of trials, number of fixations and the average number of fixations per trial. (*Figure 3.11*)

Now two terms have to be defined that are used within the following explanations.

Region of Interest (**ROI**):
Each subject's fixation just points exactly to one square in the pattern. But in order to analyze the fixation, the surrounding squares have to be included, so that

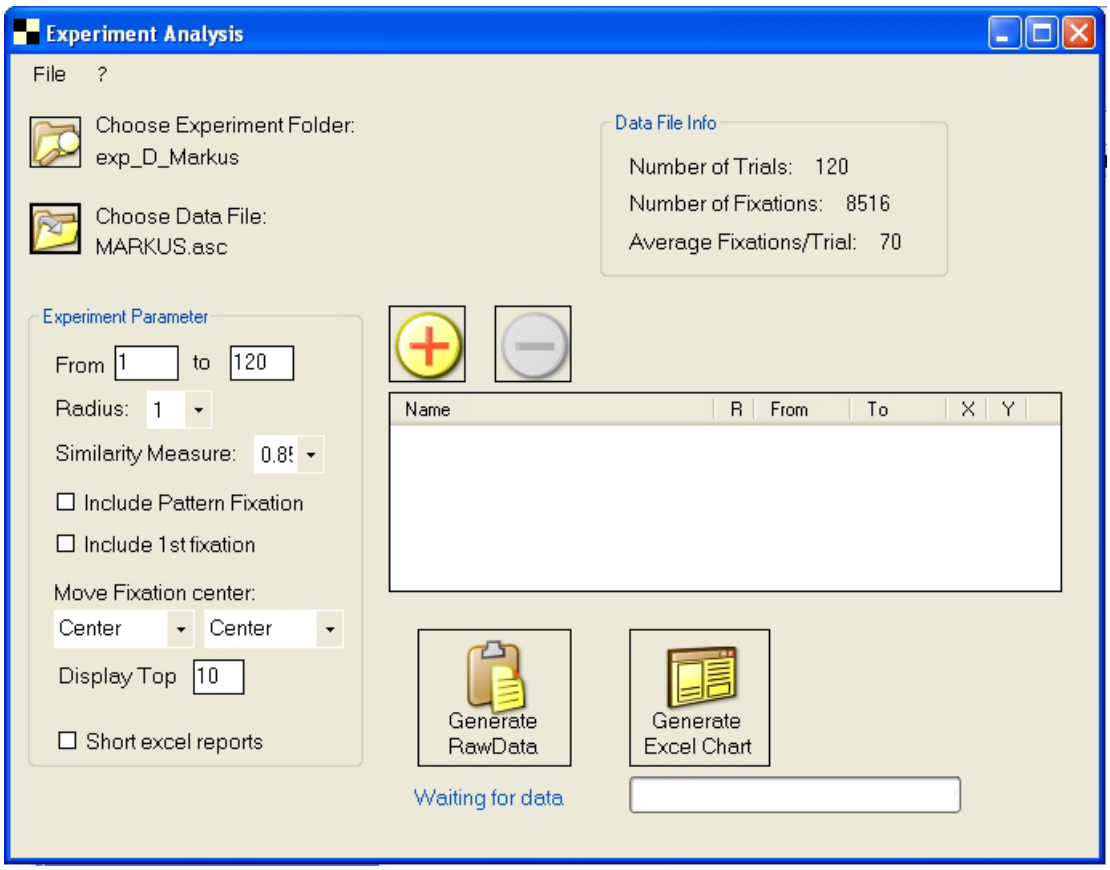


Figure 3.11: The experiment folder and data file have been selected

a region with at least the target pattern size can be used for the analysis. Hence, the surrounding squares which are needed to reach the target pattern size will create the fixation used for the analysis. This fixation area will in the following be called region of interest or **ROI**. The width and height of the target pattern builds the size of the ROI. For example, if the size of the target pattern was 3x3 squares, and the fixation was at position X:5 and Y:5 in the picture array, the area used by the analysis tool would create a ROI with the coordinates X:4 and Y:4 for the top left corner and X:6 and Y:6 for the bottom right corner. Furthermore, a radius parameter can be defined. This parameter indicates how much addititional surrounding squares (additional to the target pattern size) must be included. A radius of 0 does not increase the ROI. For a radius of 1 the ROI area would be increased: a fixation at X:5 and Y:5 would lead to a ROI starting with X:3,Y:3 top left and X:7,Y:7 bottom right. Hence, the value of the radius is added to the used target pattern size and influences the fixation area used for the calculations.

Fixation pattern of interest (**FPOI**):
Each ROI used for the analysis can be divided into one or more patterns with

size of the target pattern. E.g. for a given target pattern with 3x3 squares, and a radius parameter set to 0, the size of the ROI would be 3x3 squares and would be the same as the size of the target pattern. (In this case the fixation pattern of interest would be the same as the ROI.) For a radius of 1, the ROI would have a size of 5x5 squares, and would contain 9 patterns with the same size as the target pattern. In this case, of each ROI all patterns are extracted which have the same size as the target pattern. These extracted patterns are used for further calculations. Each of these sub-patterns extracted from a ROI will be called fixation pattern of interest or **FPOI**.

Now the user can define the parameters for the reports that are to be created by the Experiment Analysis Tool.

- *From - To*
  Defines at which trial number the analysis should start and end.

- *Radius*
  The used radius for creating the ROI.

- *Similarity Measure*
  A percentage value that describes when two FPOIs are meant to be similar. A FPOI can be compared to another pattern if both have the same size (e.g. both are 3 squares wide and 3 squares high). The number of matching colors at each position of both fixations can be expressed as a percent value - e.g. if both fixations in a 3x3 pattern are the same, but only differ in the color in their bottom right corner, the similarity of one pattern with the other would be 0.88 (or 88%). For the algorithm used in the analysis a list with all possible patterns that are similar to all FPOIs is created. The generated reports present the most frequent patterns inside this list. This list is called *high similarity list*.

- *Include Pattern Fixation*
  If this option is checked, ROIs whose surrounding squares include the target pattern are processed in the analysis. Otherwise, these ROIs are left out. By default this is not checked. Analyzing fixations that include the target pattern could lead to altered results.

- *Include 1st Fixations*
  If this option is checked, the first subject fixation in the picture for each trial is processed in the analysis. By default, this is not checked. At the start of each trial the first fixation is based on only a coarse assessment of the new scene and therefore is not fine-grained enough for the purpose of the current study.

- *Move fixation center*
  By default, a fixation is supposed to be the center of the ROI. For building the ROI, the surrounding squares depend on the target pattern size and the radius. With this option the center of the ROI can be moved by 1 square to all eight possible directions (left, top left, top, top right, right, bottom right, bottom, bottom left). E.g., to create a ROI the squares which are surrounding the center are used. With this parameter, the center can be moved either horizontal, vertical or diagonal. So if the center is moved to the top-left, the squares which are right and below the subject's fixation are used to built the ROI.

- *Display Top*
  At the end of each trial block, several rankings are displayed (e.g. the FPOIs ranked by their frequency). A detailed discussion of these rankings are found in *section 3.3.2*. This value defines for all rankings how many of the top ranks are listed in the analysis.

- *Short Excel Report*
  By default, the Excel report sheets display a list with fixation details, including their position and a graphical representation of their ROI. Checking this option will leave out the fixation details and print only the rankings.

After all parameters are set (or left to their default values) the experiment can be added to the analysis queue by clicking the *"Plus"* button. Once added, it can be selected and then removed from the queue with the *"Minus"* button.

The desired amount of experiments that should be analyzed can be added to the queue, by repeating the steps above. After all experiments are in the queue, the report creation can be started. The user can choose between generating an Excel sheet or raw data files (which are text files). (*Figure 3.12*)

The processing of all experiments can take a while, depending on the amount of trials and colors. In the following the output files are discussed in detail.

3.3.2 Analysis Output Files

The Experiment Analysis Tool can create two different output files: Excel sheets or raw data (which means text) files. In the following both outputs are described in detail. Beforehand the term "Vector" has to be explained, because the following explanations are referring to it.

Vector:
A vector is a one-dimensional representation of a (two dimensional) pattern. If the pattern is e.g. a 3x3 array of different color values, a vector of this pattern

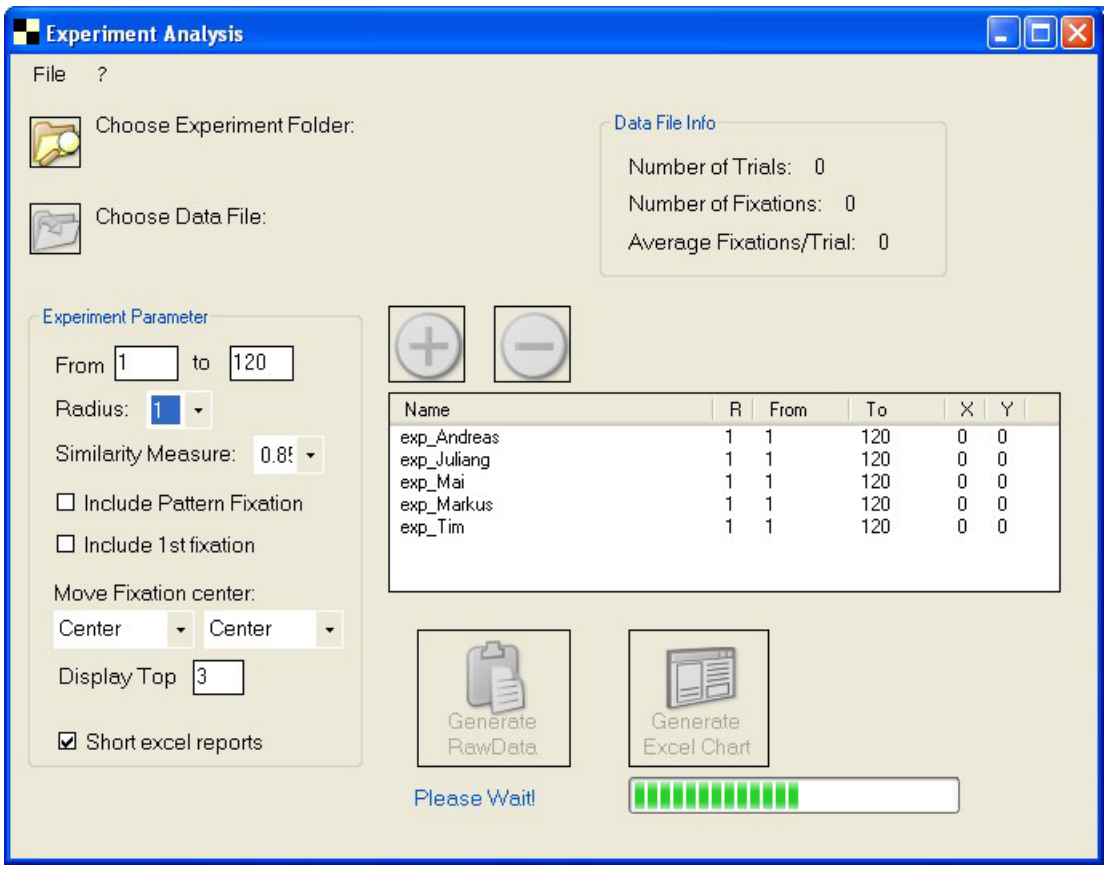


Figure 3.12: A queue was generated and the experiment analysis started

would be an one-dimensional array with the length 9. The first three color values of the vector would be identical with the top three color values of the pattern array. (The top left, top center and top right colors.) The second group of three would correspond to the colors in the middle row, and the the last three values would be the ones in the bottom row. Therefore, the conversion is row-wise from top to bottom. A vector can be written as a string by writing the abbreviated color letters in a single row (e.g. ”WWWB” is a vector string for describing a vector with the color sequence white,white,white and black).

Furthermore, a vector has the properties ”frequency” and ”weight”. If a set of vectors are combined in a list, the frequency describes the amount of occurrences in this list, and the weight any assigned value. This list can now be either sorted by frequency or weight. (In the following the value assigned for the weight is always the duration of the subject’s fixation.)

It is possible to calculate the average vector of any number of given vectors. To calculate an average vector, first the frequency of each color at the first position in every vector is counted. Then the color with the highest amount of

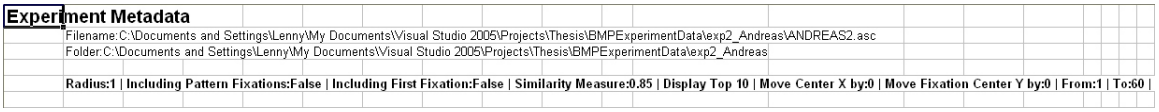


Figure 3.13: Experiment Metadata

occurrences is written at the first position of the average vector. This will be repeated for every position.

An example for building the average vector: Three vectors with two different colors and a pattern size of 2x2 squares are given. The first vector is ”WWWB”, the second ”WWBB” and the third ”WBBB”. The average vector would be ”WWBB”. For the first position the frequency for the color ”W” (white) is 3, and for ”B” (black) 0. For the second position it is for ”W” 2 and for ”B” 1, for the third position ”W” 1 and ”B” 2 and for the forth position ”W” 0 and ”B” 3. Hence, the color with the highest frequency is taken into the average vector.

Also the average vector can be built using the weight: Now not the frequency decides which color is taken, but the color with the highest sum of weight defines the new average vector. E.g., if there are four vectors ”BBWW”, ”BWWW”, ”BWWB” and ”WBBB” with their corresponding weights 50,75,100 and 200, the average vector would be ”BBWB”.

If there is no color with the highest frequency or the highest weight (because two or more colors have an equal value), then the color which has the highest value in a given color distribution is taken. The color distribution is a parameter which is always provided when calculating the average vector, normally this is the overall statistical average color distribution of all FPOIs. E.g., for two colors a color distribution could be 57% for white and 43% for black.

Two vectors can be compared to each other. The result of such a comparison is a percentage value that indicates how many positions in the two vectors contain matching colors. Comparing the vectors ”WWBB” and ”BWBW” would return the value 0.50 or 50%.

Excel Output File

The Experiment Analysis Tool will generate a new Excel sheet for each new block in the experiment. In the following the content of such an Excel sheet is discussed. Examples for complete results are found in the ”BMPExperimentData” folder on the enclosed CD.

The first two lines with the headline *”Experiment Metadata”* locate the data file and experiment folder of the output. The subsequent line describes the parameter set of the given output. (*Figure 3.13*)

Now a detailed output of data for each trial is given. (This output is only displayed if the *Short Excel Report* parameter is not set.) At the beginning of each

Trial No.:			1		
Name:			2 Colors small resolution		
Block No.:			1		
Timeout:			TRUE		
Duration of Trial:			62667		

Figure 3.14: Trial information

Type	PosX	PosY	Sq.Y	Sq.X	ViewX	ViewY	Dur.	Pup.Size	-	Distr.	Avg.Vector.	Vector Comp.			X/Y	30	31	32
Pattern	31	32	3	3	30	31				W 67% B 33%	WBWWBWBWW				31			
															32			
															33			
FIX(1st)	19	20	5	5	17	18	620	1956		W 22% B 78%	BBBBBWBBW	0.44	0.44	0.55				
												0.55	0.44	0.66				
												0.55	0.55	0.44				
FIX	18	17	5	5	16	15	772	1857		W 22% B 78%	WBBWBBBBB	0.66	0.11	0.66				
												0.66	0.33	0.44				
												0.22	0.55	0.66				
OUTSIDE	0	9																
PFIX	32	33																

Figure 3.15: Details for four different fixation types, including ROI and target pattern

trial, the trial number, name of the trial, block number the trial is part of, the information whether this trial was timed-out or ended by the subject via button press and overall duration of the trial in ms is displayed. (*Figure 3.14*)

For each trial, the **ROI** (region of interest, an area surrounding the fixation and used for the analysis) and additional details of a subject's fixation is printed. The size of the ROI depends on the used radius parameter. The ROI can be split up in different patterns with the size of the target pattern, the **FPOI**s (fixation patterns of interest, which are patterns with the size of the target pattern and are extracted from the ROI). For a radius of 0, this would be equal to the printed ROI. For a higher radius the number of FPOIs is higher.

These details include: (*see also figure 3.15*)

- *Type*
  Has one of these values:
  - "Pattern" to identify this not as a fixation but as the description of the target pattern.
  - "Fix" for a normal fixation inside the picture.
  - "FIX(1st)" to identify this fixation as the first fixation of the trial.
  - "PFIX" indicates that the ROI includes the target pattern.
  - "OUTSIDE" for a fixation on the screen outside the picture.
- *PosX/PosY*
  Describes at which square in the search picture array the subject's fixation

was in horizontal/vertical direction, starting with 0. The maximum value is the size of the picture array in horizontal/vertical direction minus one.

- *SquaresX/SquaresY*
  The width and height of the ROI, with the fixation position used as center (if not differently defined in the parameters by the *MoveX/MoveY* fields). This is the target pattern width/height when a radius of 0 is used and more if the radius is higher. In the current state of the Analysis Tool only quadratic ROIs are used, therefore width and height of this rectangle is always equal.

- *ViewX/ViewY*
  At the end of the line the ROI is printed. The values for ViewX/ViewY are coordinates that describe the position of the top left corner of the ROI in the picture.

- *Dur.*
  The duration of the subject's fixation in ms.

- *Pup.Size*
  The subject's pupil size in pixel. The value strongly depends on the distance between the camera and the subject's eye. The pupil size can describe the attention - if the subject pays a lot of attention to the current trial, the pupil size will get bigger, if not, it will get smaller. (*See [6]*)

- *Distr.*
  The color distribution of all colors in the current ROI. It displays an abbreviation for the color and a percentage of the color distribution.

- *Avg. Vector*
  This vector string describes the calculated average vector of all FPOIs that can be extracted from the current ROI. If the ROI has the same size as the target pattern (that means the radius parameter is 0), the average vector is equal to the ROI.

- *Vector Comp.*
  The vector comparison value is a percentage which describes how many positions in the average vector are matching the target pattern.

- *Printed Pattern*
  A graphical representation of either the target pattern or the ROI is printed at the end of each line.

At the end of the list that describes the ROI and fixation details for a trial block (where each Excel sheet contains exactly one trial block) an analysis with color distributions data and different rankings follows.

Overall Color Distribution by Bins					
		Color W			
			Rank	Percent	Frequency
			1	56%	9306
			2	44%	8522
			3	67%	5783
			4	33%	5077
			5	78%	2286
			6	22%	1759
			7	89%	451
			8	11%	387
			9	100%	33
			10	0%	32
		Color B			
			Rank	Percent	Frequency
			1	44%	9306
			2	56%	8522
			3	33%	5783
			4	67%	5077
			5	22%	2286
			6	78%	1759
			7	11%	451
			8	89%	387
			9	0%	33
			10	100%	32

Figure 3.16: Color distribution example

Average vector of all Fixations:	W 0% B 100%	BBBBBBBBB

Figure 3.17: Average vector of all FPOIs

First of all, the parameter set is printed again. Then the overall statistical color distribution for all ROIs is given. After this comes a detailed list of the color distribution, separated into bins and ordered by their frequency, related to the size of the target pattern. This is because the color distribution has always discrete values, e.g. for a 2x2 target pattern, the color distribution is either 0%, 25%, 75% or 100% for each color. The distribution list ranks the frequency of each color for each possible percent value, related to the pattern size and the number of colors. (*Figure 3.16*)

In the next part of the rankings the average vector is built, combining all FPOIs which are extracted out of the ROIs for the analysis. The average vector is displayed as a vector string (the abbreviated color letters in a single row), followed by its color distribution and a graphical representation of this vector. (*Figure 3.17*)

Next there are several "Top-**X**"-Rankings. **X** is a number defined in the parameter set and describes how many of the first patterns in the rankings are displayed. It starts with a ranking that lists the top **X** of all FPOIs, and it is called "fixation ranking". All FPOIs are here ranked by their frequency. (*Figure 3.18*)

Each ranking starts with a repetition of the target pattern. Then the ranks for the first places are displayed, including the vector string of each pattern

			Top 10 of fixations, ranked by frequency					
		Pattern:		W 67% B 33%	WBWWBWBWW			
			Rank	Color Distr.	Vector	Frequency	Weight	Similarity
			1	W 56% B 44%	BBWWBWBWW	157	4553	0.88
			2	W 44% B 56%	WBWBWBBWB	149	4768	0.55
			3	W 56% B 44%	WBWWBWBWB	141	3243	0.88
Average Vector				W 56% B 44%	WBWWBWBWB			0.88
Average Vectorusing Weight				W 56% B 44%	WBWWBWBWB			0.88

Figure 3.18: Top 3 of a fixation ranking

in the ranking, as well as the color distribution of this pattern, the amount of occurrences (frequency) in the ranking, the calculated weight and a graphical representation of this pattern. In all rankings the weight is the total time of the fixation duration, inherited from the ROI from which the FPOI was extracted. The similarity is a percent value and describes how similar the described pattern is to the target pattern.

After the third and each fifth place an average vector of all precedent ranking vectors is built and displayed. This means that all previous patterns and their vectors are combined to an average vector by counting the frequency of each color for each position in these previous patterns, and keeping the color with the highest frequency at each position in the vector. If the frequency for a position is the same for all colors, the color which has the highest value in the overall color distribution is taken. An average vector using weight means that the weight of each vector is added up for each color instead of the frequency - the color with the highest weight is taken into the average vector. (*Figure 3.18* shows an example for the average vector in a ranking.)

The same ranking is generated for all possible patterns that have a ”high similarity to the target pattern.” That means that for each FPOI all patterns are calculated which are similar to this FPOI, depending on the defined similarity parameter. Then the patterns are added to the high similarity list and the same procedure is repeated with the next FPOI. E.g., if the similarity value is 0.80, and the target pattern size 3x3, the high similarity list would contain all patterns which are equal to each FPOI but differ in only one color. Therefore, the ranking

Top 10 patterns with high similarity to target pattern, ranked by frequency					
Pattern:	W 67% B 33%	WBWWBWBWW			
Rank	Color Distr.	Vector	Frequency	Weight	Similarity
1	W 67% B 33%	WBWWBWBWW	837	23436	0.99
2	W 44% B 56%	BBWWBWBWB	824	63448	0.77
3	W 56% B 44%	WBWBWBBWW	793	18239	0.66
Average Vector	W 67% B 33%	WBWWBWBWW			0.99
Average Vectorusing Weight	W 44% B 56%	BBWWBWBWB			0.77

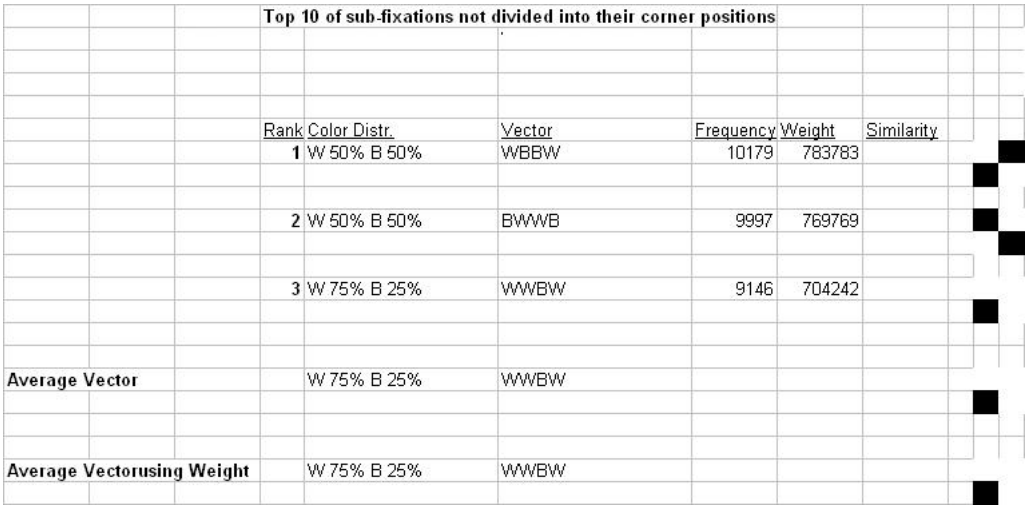
Figure 3.19: Example for the high similarity ranking

displays the most frequent patterns in the high similarity list. In difference to the fixation ranking this ranking can contain the target pattern. (*Figure 3.19*)

Also a ranking by a defined score system is created. This ranking takes every pattern inside the high similar list and compares it again with each possible pattern. If the similarity is higher than 60% (a fixed value), the pattern will get a score, depending on the similarity to the possible pattern. E.g., if both of them have in 7 out of 9 positions the same color, the similarity value would be 77, and this value would be added to the score of the pattern. This ranking was implemented to see if it would give interesting results, but did not fulfill the expectations and is not regarded in further discussions.

The next ranking shows a placement of the ”sub-fixations”. A FPOI is split up into 4 smaller patterns, which are in the following called ”sub-fixations”. A sub-fixation is one square smaller in width and height than the FPOI. E.g., given a FPOI of 3x3 squares, the size of a sub-fixation would be 2x2. Each one of the four sub-fixations starts at a different corner of the FPOI (top left, bottom left, top right, bottom right) whereby the sub-fixations overlap. The sub-fixation lists can now be ranked by their frequency. There are five rankings for the sub-fixations: four for the different corner positions, and one where all sub-fixations are not separated by their corner position. *Figure 3.20* shows an example for the overall sub-fixation ranking.

The next ranking is an approach that tries to combine the sub-fixations for each corner position into a new ranking. Place one in this ranking is created by taking the top-position of each corner, and putting them together into a pattern



Top 10 of sub-fixations not divided into their corner positions

Rank	Color Distr.	Vector	Frequency	Weight	Similarity
1	W 50% B 50%	WBBW	10179	783783	
2	W 50% B 50%	BWWB	9997	769769	
3	W 75% B 25%	WWBW	9146	704242	
Average Vector	W 75% B 25%	WWBW			
Average Vectorusing Weight	W 75% B 25%	WWBW			

Figure 3.20: Example for the overall sub-fixation ranking

with the target pattern size. At each position where a sub-fixation overlaps with another sub-fixation, the sub-fixation with the highest frequency value is "on top". This approach tried to combine the sub-fixations into possible patterns that would be close to the target patterns, but did also not meet the expectations and is not regarded in further discussion.

After this the same rankings are repeated, but this time not ordered by frequency but by its weight - which is the duration of each subject's fixation.

Raw Data Output Files

The raw data files require a folder, which the user specifies when he clicks the generate button for the raw data. Inside this folder, for each block of trials three different files are created. For each file, the **X** marks the block number, starting with 1.

- `lookupX.txt`
  A so called look-up file, which creates all possible patterns, depending on the height, width and number of colors of the target pattern of a trial block. For each possible pattern an ID is given, starting with 1 for the first possible pattern, counting up. This way each possible pattern that can be created has an unique ID.

- `rawDataX.txt`
  For each FPOI, the previously created ID of this pattern is written in each line, followed by the color distribution for each color. At the beginning of each block, a java style comment with the parameter set is printed. At the

beginning of each trial, a java style comment with the target pattern ID is printed.

- `resultX.txt`
  This file contains a text style version of the fixation and high similarity rankings as described in the Excel output.

The goal of the raw data files was to use a Data Mining Tool such as kNime (*see [8]*) to analyze the data in an easy way. Unfortunately, the data analysis with kNime did not lead to valuable results and is not regarded in further discussion.

3.3.3 Class Overview

In the following a brief overview of the classes used for the Experiment Analysis Tool is given. The Experiment Analysis Tool consists of two parts: A class library that represents the different eye events of the generated `asc` data file from the EyeTracker, called "ASCLibrary". The other part contains classes for the GUI and the generation of the output files from the Experiment Analysis Tool and is called "BMPAnalysis".

The following section gives a rough overview of the code structure and its classes. Additional to the structural overview the complete code documentation can be found in *appendix C* for the Experiment Analysis Tool and *appendix B* for the ASCLibrary . The complete source code, the project files and its documentation in electronic form is located on the enclosed CD in the folders "ASCLibrary" and "BMPAnalysis".

The Experiment Analysis Tool is written in C#, using the .NET 2.0 Framework from Microsoft (*[10]*). It was developed with Visual Studio 2005. A list of all used software tools is listed in *appendix A*.

ASCLibrary Class Overview

The `ASCLibrary` is a representation of an EyeTracker data file. It is a library that provides the possibility to access the eye tracker data and events stored in the `asc` file. Each `asc` file contains data information for each trial run during the experiment. The `ASCData` object represents a list of trials of the type `ASCTrialData`, and provides a method to load all trials from the file into this object. Each trial is therefore represented by an `ASCTrialData` object that contains its meta data and a list of all fixations, saccades and blinks of the subject's eye. The fixations, saccades and blinks are called events. Each fixation is stored in an `ASCFixation` object, containing start and end time in ms relative to the start of the experiment, the duration, the average pixel coordinates of the fixation on the screen and the average pupil size in pixel during the fixation. A saccade and its `ASCSaccation` object describe the start and end time of a saccade,

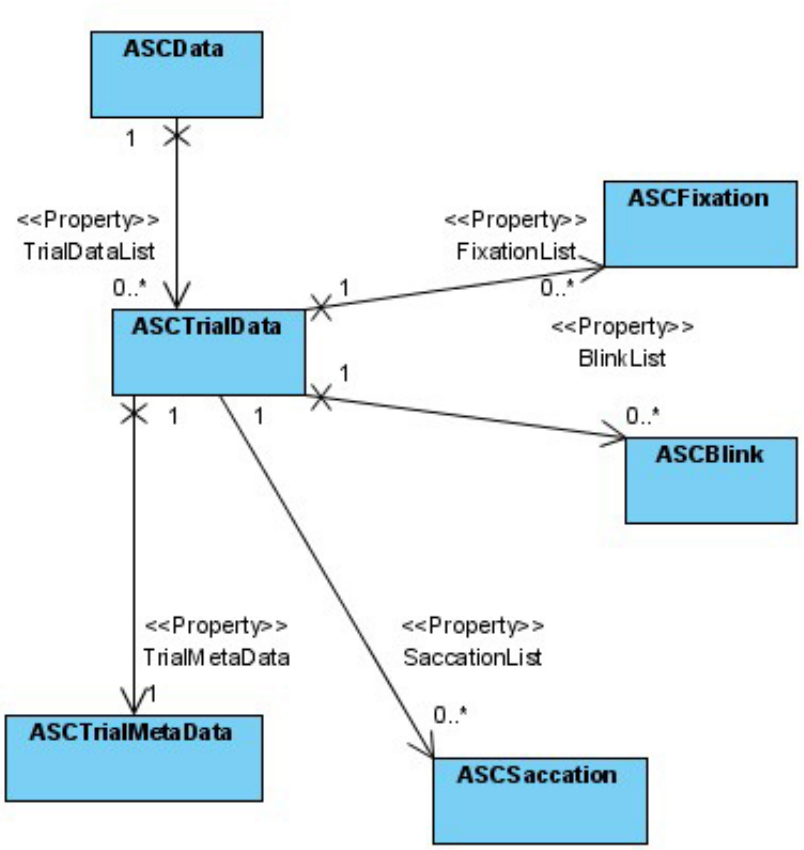


Figure 3.21: Class diagram of ASCLibrary

its duration, the start and end coordinates on the screen, the saccadic amplitude and the peak velocity of a saccade. (The made-up word "Saccation" was used unintendedly.) A blink event and its `ASCBlink` object represent a blink of the eye and contain the start and end time as well as its duration of the blink. The correlation of the objects is shown in the class diagram in *figure 3.21*.

Experiment Analysis Tool Class Overview

The Experiment Analysis Tool consists of objects for creating the GUI, namely `MainForm` and `Info` (*figure 3.22*). `Program` simply invokes the main window `MainForm`, and `Info` displays version and author information. The `MainForm` object represents all functions as described in *section 3.3.1*. The entered information, specifically the list of experiments to be analyzed and its parameters, folder and data file, are stored in an `AnalysisObject`. This object contains the list of parameter in its property `OutputFileParameter`, and some spe-

cific experiment data in the `Config` object (this is the representation for the file `config.txt` in the experiment folder). Furthermore, the analysis objects use the existing `Bitmap` object from the Experiment Builder (*see 3.1.3*) to load and access the experiment data, as well as the `ASCLibrary` to load and access the data stored in the eye tracker data file.

A list of the current analysis objects in the queue is stored and displayed by the GUI. When the user starts one of the file generation processes (either Excel sheets or raw data), the appropriate child of the `OutputFile` object is generated and fed with the necessary parameters and experiment data. `OutputFile` is an abstract object - depending on the type of output desired, either an `OutputFileExcel` or `OutputFileRawData` object is instantiated. `OutputFile` implements all necessary calculations for the analysis of the experiment data - they are the same for both output types. For the calculations several `Vector` objects are used, which represent patterns and are handy to do calculations within these patterns. (*A description of a vector is given in 3.3.2.*) Also the `ColorDistribution` object is used to help with color distribution calculations. `OutputFile` provides so called hook-up methods which are used by either `OutputFileExcel` or `OutputFileRawData` to display the calculation results.

Hence, the calculation algorithms are implemented in the `OutputFile` object. The result of these algorithms are displayed within the rankings in the output files created with `OutputFileExcel` or `OutputFileRawData`.

A discussion of a conducted experiment and its results is given in the next chapter.

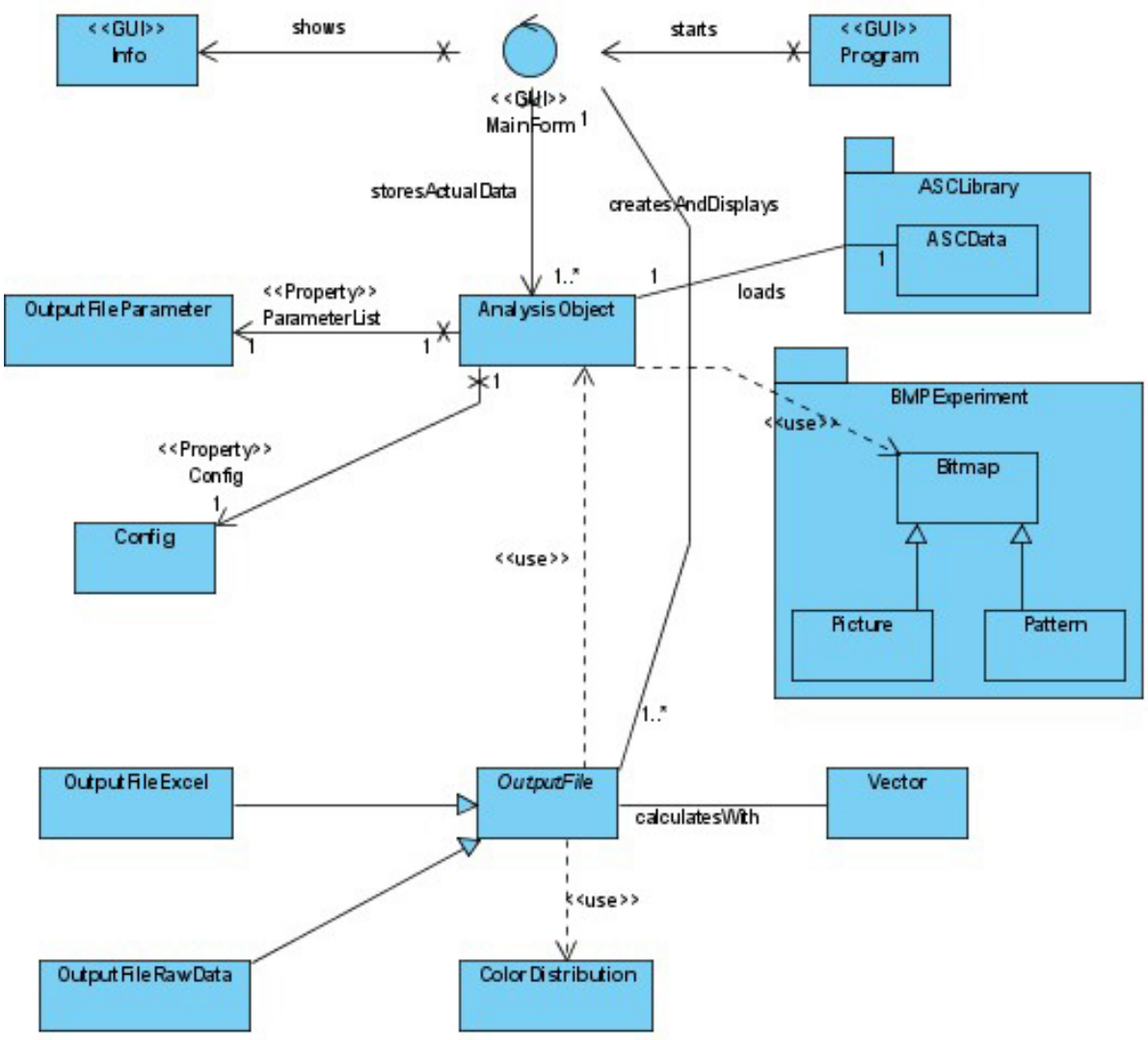


Figure 3.22: Class diagram of the Experiment Analysis Tool

Chapter 4

Experiment Setup, Run and Analysis

4.1 Experiment Setup

4.1.1 Participants

This research was carried out with the assistance of 14 participants of ages 19 to 36, divided into two groups for different experiment setups. Participants were paid a $10 honorarium. One of the participants was the author, two were lab members. The rest was naive as to the purpose of the experiment. All of them had either normal or corrected-to-normal vision. There were two different experiment setups - 5 participated in the first setup, 9 in the second.

4.1.2 Apparatus

Stimuli were presented on a DTI Virtual Window 19" TFT-Display. The resolution of this display was set to 1280x1024 pixels and its refresh rate to 60 Hz. Participants sat approximately 60 cm from the screen. The horizontal and vertical viewing angles of the stimuli were approximately 34 degrees and 26 degrees, respectively. An SR Research EyeLink II system was used to track eye movements. The average error of visual angle in this system is 0.5 degrees, and its sampling frequency was set to 250 Hz. A game-pad was used to register the participants' manual responses.

4.1.3 Procedure

The experimenter began each experiment by providing the participant with task instructions, fitting the eye tracking headset to the participant, and then cal-

ibrating the EyeTracker. Participants started each trial, and at the same time performed a drift correction of the headset, by pressing a button while fixating on a central marker. Each trial began with the presentation of a target pattern for about 6 seconds, during which the participants were to memorize the target structure. After three trials, the display time was reduced to 3 seconds, and after 15 trials to about 2 seconds. Subsequently, the search picture appeared. Participants were to search the picture array for the target pattern and, while fixating on the pattern, to press a button on a game pad to terminate the trial. If a participant did not press the button within a specific time interval after search display onset, the trial would time out and the experiment would continue with the next trial. After termination of each trial, a red box framed the actual target pattern position, so that feedback to the participants about their accuracy was provided.

The pattern and the picture array consisted of squares in either two or three colors combined to a random-dot array. The distribution of the squares was random, while for 2 colors the possibility of appearance for each color was 50% and for 3 colors 33% each. (*See figure 2.7 and figure 2.8 for examples.*)

There were two different experiments: The first included 4 blocks of trials, the second 2 blocks of trials. In both each block contained 30 trials. For the first experiment, in the first two blocks the size of the squares were 60x60 pixel, and in the last two 40x40 pixel. The first and third block had squares in 2 colors, and the second and last block in 3 colors. For the second experiment both blocks contained squares with a size of 25x25 pixel. The first block was with 2 colors, the second block with 3 colors. The background color for both experiments was always white.

For each block of trials, always the same pattern was used. This means that a subject had only to memorize 4 different patterns for experiment I (two with 2 colors and two with 3 colors) and only two different patterns for experiment II (one with 2 colors and one with 3 colors). In order not to bore the subject, the display time of a pattern was longer for the first three trials (6 seconds), and then reduced (3 seconds). After 15 trials it was reduced again (2 seconds), because the subjects now for sure memorized the pattern. By using the same pattern for a block of trial, the possibility that a subject would not be able to memorize the target pattern was excluded.

It follows a detailed parameter list for both experiment setups:

Experiment I:

- **Trial block 1** "2 Colors big"
  Square size 60x60 pixel, 2 colors, 30 trials, array size: 3x3 for target pattern, 17x17 for search picture, maximum duration 35 sec.

- **Trial block 2** "3 Colors big"
  Square size 60x60 pixel, 3 colors, 30 trials, array size: 3x3 for target pattern,

17x17 for search picture, maximum duration 35 sec.

- **Trial block 3** "2 Colors small"
  Square size 40x40 pixel, 2 colors, 30 trials, array size: 3x3 for target pattern, 25x25 for search picture, maximum duration 40 sec.

- **Trial block 4** "3 Colors small"
  Square size 40x40 pixel, 3 colors, 30 trials, array size: 3x3 for target pattern, 25x25 for search picture, maximum duration 40 sec.

Experiment II:

- **Trial block 1** "2 Colors big"
  Square size 25x25 pixel, 2 colors, 30 trials, array size: 3x3 for target pattern, 40x40 for search picture, maximum duration 60 sec.

- **Trial block 2** "3 Colors big"
  Square size 25x25 pixel, 3 colors, 30 trials, array size: 3x3 for target pattern, 40x40 for search picture, maximum duration 60 sec.

Both experiment setups are on the enclosed CD. ("BMPExperimentData/template1.bin" and "BMPExperimentData/template2.bin", to be opened with the Experiment Builder, *see section 3.1*.)

4.2 Experiment Runs

The first experiment was run with 5 subjects. An experimenter could see the eye movements of the subject on the eye tracker display screen. The EyeTracker data files were saved and later analyzed with the Experiment Analysis Tool (*See section 3.3*). After a first brief analysis of the results and observation of the subjects during a trial it was discovered that the number of fixations for each block of trials was relatively low, according to the "big" square size and the easiness for a subject to find the target pattern (subjects were able to find the target pattern very quickly). Also, they just started to scan the screen in a systematically way, e.g. line by line in reading direction. Therefore the overall fixations could have consisted of a more or less evenly distributed list of fixations on the screen.

The smaller the size of the squares, the more subjects tended to "jump" with their fixation randomly across the search picture - it seemed that for a smaller square size the subjects scanned "unsystematically" for the pattern. Another observation for experiments with a smaller square size was that the pattern was harder to find and more fixations were needed to find it - also it could be observed that the subjects looked more often at already processed parts of the image, which was a sign for an unsystematical scanning for the target pattern.

Because of this the second experiment was run with 9 subjects, in the assumption it would deliver more fixations and more ”intuitive” eye movements of the subject. Though the results could be more inaccurate because of the Eye-Tracker inaccuracy, the second experiment had a higher number of subjects' fixation and seem to provide more useful empiric data for the analysis of the experiment.

4.3 Experiment Analysis

For each subject, excel files were created with the Experiment Analysis Tool. (*See section 3.3.1*) The following parameters were applied:

- Similarity measure: 0.85
- Radius 0 and 1
- All trials
- Pattern fixations not included
- First fixations not included
- Fixation center is in the center of the pattern

A list of all created reports that contain the analysis for experiment I and II are on the enclosed CD in the subfolders of ”BMPExperimentData”, where every folder starts with the prefix ”exp_NAME” for the first experiment and ”exp2_NAME” for the second experiment. The Excel reports are located under ”xls”. The filename contains the subject's name, the block number and the used radius. The short versions of these reports include the word ”short”.

The reports were generated for a radius of 1 and 0, therefore there are statistics for each radius.

4.3.1 Reports Experiment I

The target patterns for each trial block of the first experiment are named with the letters A to E for each of the five subjects and 1 to 4 for each trial block. *Figure 4.1* shows the patterns A1-A4, *figure 4.2* B1-B4, *figure 4.3* C1-C4, *figure 4.4* D1-D4 and *figure 4.5* E1-E4. The size in this pattern figures is relative to the size during experiment display. However, in further figures the patterns are displayed in a unique size.

Figure 4.6 shows an example of the recorded fixations for subject E (trial no. 67) superimposed in green. The target pattern E3 is marked in red.

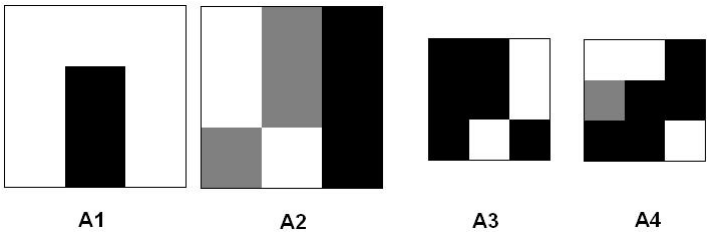


Figure 4.1: Patterns A1 to A4

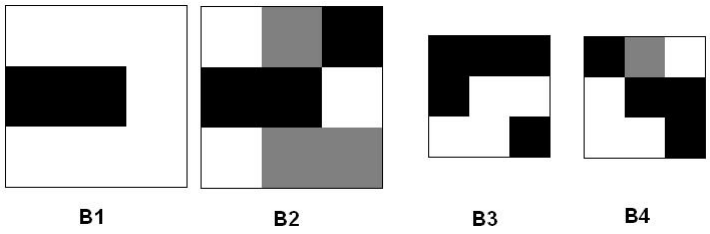


Figure 4.2: Patterns B1 to B4

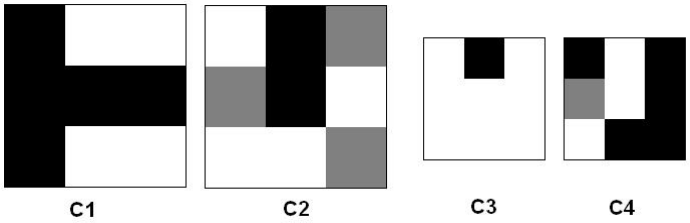


Figure 4.3: Patterns C1 to C4

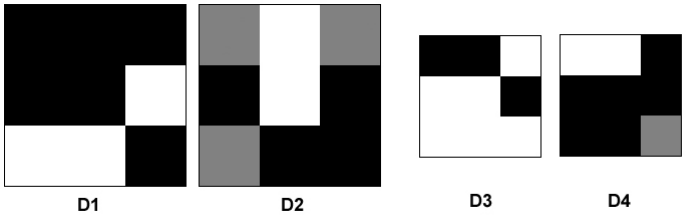


Figure 4.4: Patterns D1 to D4

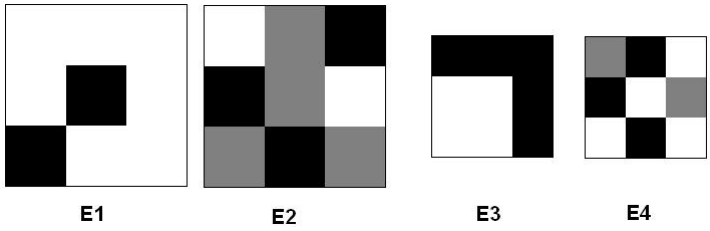


Figure 4.5: Patterns E1 to E4

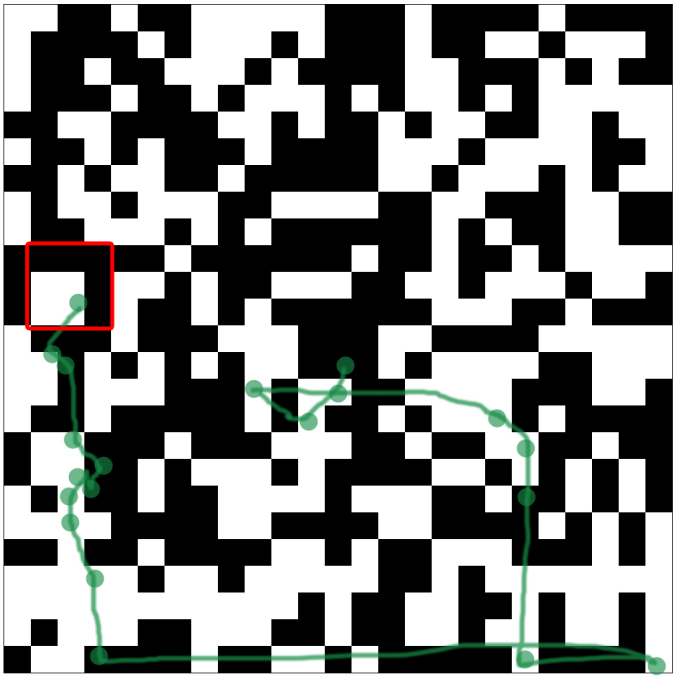


Figure 4.6: Fixations superimposed in green, target pattern E3 in red.

Average Vector and Color Distribution

The average vector of all FPOIs (a pattern that describes the most frequent color for each position in all FPOIs) was build for each trial block. Also the overall color distribution was calculated.

For the trials with 2 colors the average vector was in nearly all cases a unique white or black pattern. Whether it was black or white depended on the stronger value in the average color distribution. If the target pattern was more white than black (that means, 5 or more out of the 9 squares in the target pattern were white) the average vector was in most cases more white than black.

Out of this perception, the question arose if the color distribution of the target pattern can be associated with the color distribution of the FPOIs. *Table 4.1* shows the color distribution of the target pattern and the statistical average color distribution of all FPOIs for block 1, where a radius **R** of 1 or 0 was used for the calculations. *Table 4.2* displays the same overview for block 3.

Table 4.1: Color distribution block 1 (Exp. I)

Pattern	Target Pattern Distr.	FPOIs R1	FPOIs R0
A1	78%/22%	57%/43%	62%/38%
B1	56%/44%	54%/44%	57%/43%
C1	44%/56%	47%/53%	45%/55%
D1	33%/67%	45%/55%	44%/56%
E1	67%/33%	56%/44%	58%/42%
Values are white/black percentages			

Table 4.2: Color distribution block 3 (Exp.I)

Pattern	Target Pattern Distr.	FPOIs R1	FPOIs R0
A3	33%/67%	43%/57%	40%/60%
B3	44%/56%	51%/49%	51%/49%
C3	89%/11%	55%/45%	57%/43%
D3	67%/33%	54%/46%	55%/45%
E3	44%/56%	53%/47%	53%/47%
Values are white/black percentages			

Regarding the color distribution for block 1 and 3, a direct connection between the color distribution of the target pattern and the color distribution of all FPOIs can be made - if white is the stronger color in the target pattern, the overall color distribution of all FPOIs for white is higher, and vice versa.

Regarding trial blocks 2 and 4 which used 3 colors, the average pattern of all FPOIs mostly contained patterns dominated by black - it seems that black was a dominant color. The color distribution of the target pattern and the statistical average color distribution of the FPOIs are presented in *table 4.3* for trial block

2 and in *table 4.4* for trial block 4. They show that black has always the highest percentage.

Table 4.3: Color distribution block 2 (Exp. I)

Pattern	Target Pattern Distr.	FPOI R1	FPOI R0
A2	33%/33%/33%	29%/38%/32%	28%/39%/33%
B2	33%/33%/33%	30%/35%/34%	29%/36%/34%
C2	44%/22%/33%	32%/35%/32%	32%/35%/32%
D2	22%/44%/33%	31%/40%/29%	31%/41%/27%
E2	22%/33%/44%	31%/36%/33%	30%/36%/34%
Values are white/black/gray percentages			

Table 4.4: Color distribution block 4 (Exp.I)

Pattern	Target Pattern Distr.	FPOI R1	FPOI R0
A4	33%/56%/11%	30%/40%/30%	28%/42%/28%
B4	44%/44%/11%	34%/36%/29%	34%/38%/28%
C4	33%/56%/11%	31%/38%/30%	31%/39%/29%
D4	22%/67%/11%	29%/42%/28%	28%/46%/26%
E4	44%/33%/22%	35%/34%/31%	35%/34%/30%
Values are white/black/gray percentages			

Fixation and High Similarity Rankings

For each FPOI a list with patterns that are similar were built. A pattern was defined as similar to a target pattern when the similarity was higher than 85%, which means that 8 of 9 colors at each position where in both patterns the same. A ranking sorted by the frequency of all FPOIs (called **fixation ranking**) and their similar patterns (called **high similarity ranking**) were created with the Experiment Analysis Tool. For a detailed discussion of these rankings see *section 3.3.2*.

The number of possible patterns that exist for a size of 3x3 squares and 2 colors is 512. For 3 colors it is 19683. This means that the list with similar patterns may contain up to 512 patterns for 2 colors and up to 19683 patterns for 3 colors.

To display all generated rankings would exceed the scope of this work. In the following comprehensive tables are presented that attempt to summarize the results of the high similarity rankings.

Table 4.5 shows the results of the high similarity list for trial block 1, with a radius **R** of 1 or 0. The first column names the target pattern, the second and forth column the rank of the target pattern in the high similarity rankings (for different radii), and the third and fifth column the similarity of the average pattern to the target pattern (also for different radii). The average pattern was build out of the first three places. A value of 100% means it is the same as the target pattern, 88% means that 8 of 9 positions have the same color, and so on. With the average pattern the approach was to combine the most similar patterns in such a way that they would result in the target pattern. The average pattern was build for the first three, five and ten places of a ranking - the average patterns for the first three places showed the best results, therefore the similarity of this pattern to the target pattern is printed in the results. It is a good indicator how similar the top patterns in a ranking have been - a similarity of 77% (7 out of 9) or above would indicate three highly similar patterns in the top three places of the high similarity ranking.

Table 4.5: Target pattern rankings in high similarity list, block 1

Pattern	Ranking R1	Sim. AvgP. R1	Ranking R0	Sim. AvgP. R0
A1	Rank 4	66%	Rank 1	88%
B1	Rank 2	100%	Rank 50	33%
C1	Rank 76	33%	Rank 239	55%
D1	Rank 2	88%	Rank 13	55%
E1	Rank 5	55%	Rank 1	100%

A graphical example for a fixation ranking and the corresponding high similarity ranking for the target pattern D1 is displayed in *figure 4.7*. The fixation ranking (left side) contains the rank for each pattern, the frequency of this pat-

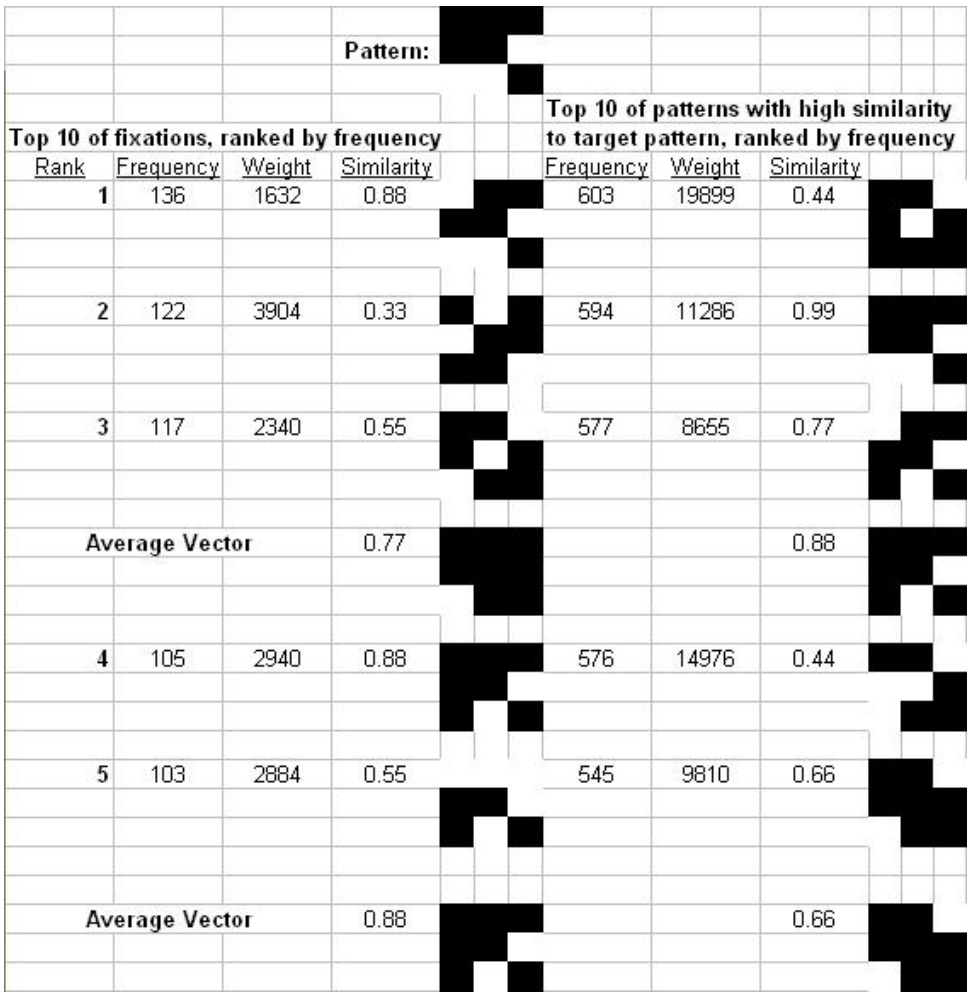


Figure 4.7: Pattern D1 fixation and high similarity ranking, radius 0

tern in the list, the weight (which is the summed duration of all fixations of this pattern), its similarity to the target pattern and the pattern itself. The high similarity list (right side) differs in a way that it contains up to eight times more patterns. The high similarity list may contain the target pattern, the fixation list not.

Table 4.6 shows a summary for trial block 3, which had like trial block 1 only two colors, but used a smaller square size. Additional to this table *figure 4.8* depicts the high similarity ranking of the target pattern D3 (used radius 0) and *figure 4.9* pattern C3 with radius 1.

Table 4.7 and *table 4.8* show a summary for trial blocks 2 and 4, which both have 3 colors and differ in their square size. The algorithm seems to work poorly for 3 colors, compared to the results for 2 colors. But the fixation and high similarity ranking for target pattern E4 (*figure 4.10*) is a good example that in some

Table 4.6: Target pattern rankings in high similarity list, block 3

Pattern	Ranking R1	Sim. AvgP. R1	Ranking R0	Sim. AvgP. R0
A3	Rank 12	66%	Rank 18	66%
B3	Rank 61	44%	Rank 105	44%
C3	Rank 3	88%	Rank 4	77%
D3	Rank 8	77%	Rank 1	100%
E3	Rank 6	66%	Rank 42	77%

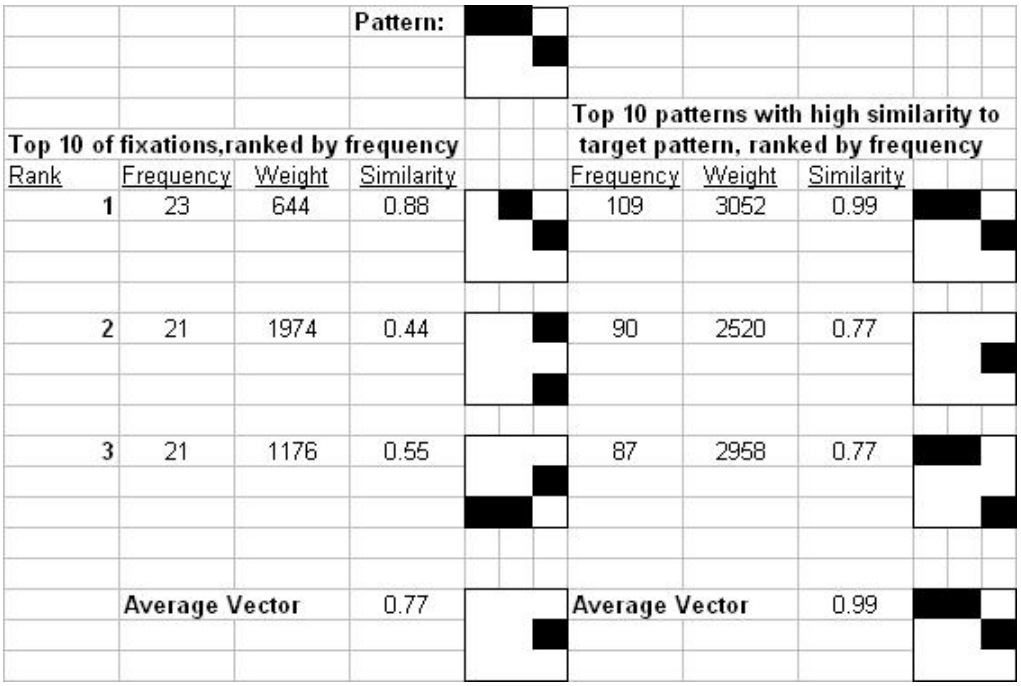


Figure 4.8: Pattern D3 fixation and high similarity ranking, radius 0

cases the basic shape is rotated or moved into one direction and can be identified.

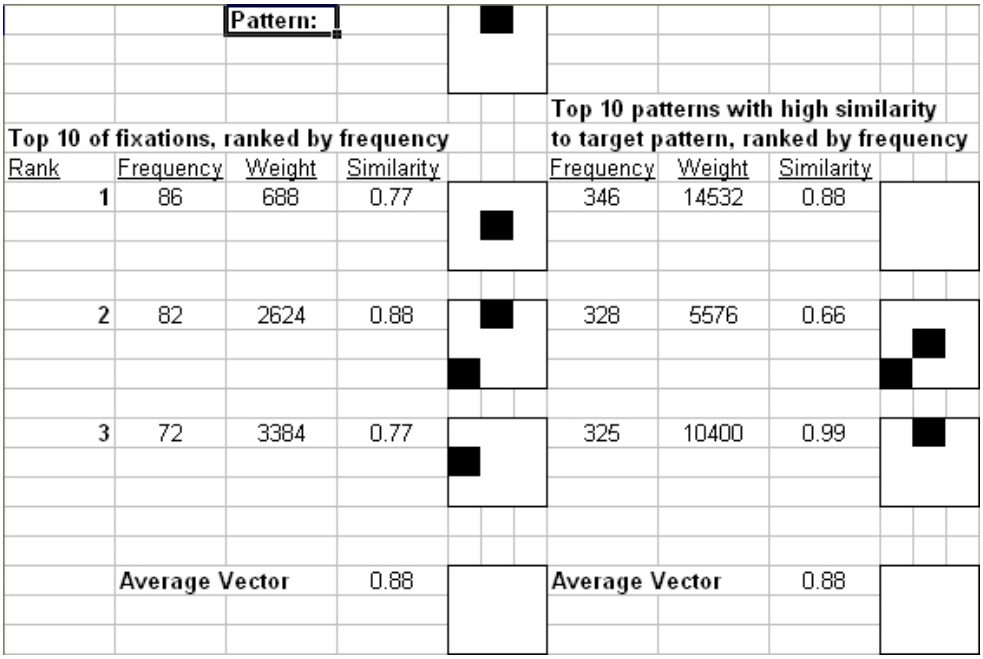


Figure 4.9: Pattern C3 fixation and high similarity ranking, radius 1

Table 4.7: Target pattern rankings in high similarity list, block 2

Pattern	Ranking R1	Sim. AvgP. R1	Ranking R0	Sim. AvgP. R0
A2	Rank 14	77%	Rank 301	55%
B2	Rank 956	33%	Rank 1111	11%
C2	Rank 1159	22%	Rank 2327	33%
D2	Rank 4683	44%	Rank 10225	33%
E2	Rank 176	22%	Rank 1413	11%

Table 4.8: Target pattern rankings in high similarity list, block 4

Pattern	Ranking R1	Sim. AvgP. R1	Ranking R0	Sim. AvgP. R0
A4	Rank 52	66%	Rank 259	44%
B4	Rank 457	77%	Rank 203	44%
C4	Rank 5163	33%	Rank 899	55%
D4	Rank 886	55%	Rank 1965	66%
E4	Rank 580	11%	Rank 412	0%

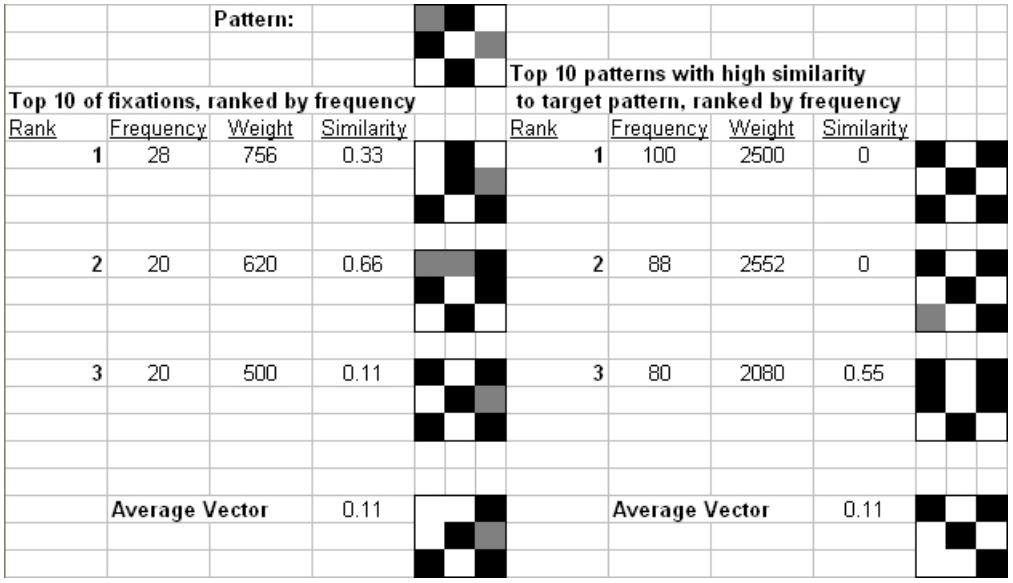
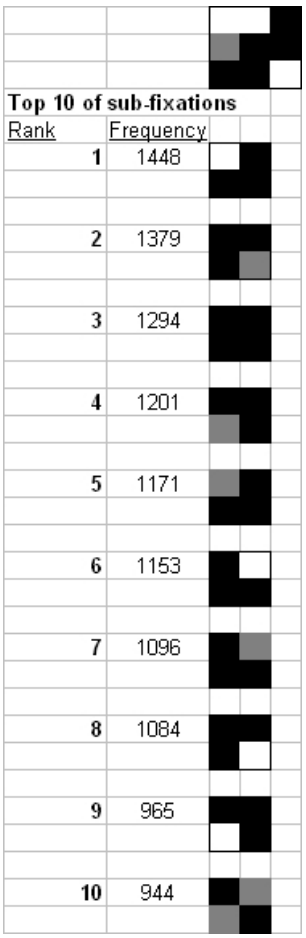


Figure 4.10: Pattern E4 fixation and high similarity ranking, radius 1



Top 10 of sub-fixations	
Rank	Frequency
1	1448
2	1379
3	1294
4	1201
5	1171
6	1153
7	1096
8	1084
9	965
10	944

Figure 4.11: Sub-fixation ranking for pattern A4, radius 1

Sub-fixations

Next, an analysis of the rankings for the sub-fixations was conducted. A sub-fixation splits up each FPOI into four 2x2 sub-patterns. E.g., the FPOIs have a size of 3x3 squares, so there are four 2x2 fixations inside this FPOI. A list containing all sub-fixations for each FPOI can be created and ranked by the frequency of a sub-pattern in the list (called **sub-fixation ranking**). The sub-fixation ranking shows the most frequent sub-fixations ranked by their frequency of occurrences in this list. *Figure 4.11* shows the Top 10 of sub-fixations for pattern A4 (used radius 1).

It is possible to separate the sub-fixations into 4 different sub-fixation rankings, each one for their position in the FPOI: a ranking for all sub-fixations of the top left corner, of the bottom left corner, of the top right corner and of the bottom right corner. But if done so, the generated list produces similar results for each corner. Compared to a list where they are not separated by their position

in the FPOI there is no big difference between them. Therefore, there is no extra information by separating the sub-fixations by their positions.

Table 4.9 shows an overview for trial block 1 and 3 (both trial blocks with 2 colors, differing in their square size) in which it is checked if the first sub-fixation in the sub-fixation ranking was part of the target pattern. If it was not part of it, the rank is printed at which the first sub-fixations in the rankings occurs that is part of the target pattern. *Table 4.10* does the same for trial block 2 and 4 (3 colors, different square size).

Table 4.9: First place in sub-fixation rankings is part of target pattern, block 1 and 3

Pattern	Block 1 R1	Block 1 R0	Pattern	Block 3 R1	Block 3 R0
A1	No (Rank 2)	Yes	A3	Yes	Yes
B1	Yes	Yes	B3	Yes	Yes
C1	Yes	Yes	C3	Yes	Yes
D1	Yes	Yes	D3	Yes	Yes
E1	No(Rank 2)	Yes	E3	Yes	Yes

Table 4.10: First place in sub-fixation rankings is part of target pattern, block 2 and 4

Pattern	Block 2 R1	Block 2 R0	Pattern	Block 4 R1	Block 4 R0
A1	Yes	Yes	A4	Yes	Yes
B1	No (Rank 2)	Yes	B4	Yes	Yes
C1	Yes	Yes	C4	Yes	Yes
D1	Yes	Yes	D4	No (Rank 2)	Yes
E1	Yes	Yes	E4	Yes	Yes

The results show that a remarkably high number of sub-fixations on first place in the rankings were part of the target pattern. The overall number of possible sub-fixations with a size of 2x2 squares is 16 for 2 colors and 81 for 3 colors. Especially for 3 colors the results are remarkably good.

Duration of Fixations

One way to rank the fixation and high similarity lists is to sort them by their frequency - another is to use the fixation duration which can be used as a weighting factor for a ROI and the extracted FPOIs. (*See section 3.3.2 for details about using the fixation duration as weight for a ranking.*) An examination of rankings that use the duration of the eye fixation as a weight and sorting factor for the ranking instead of the frequency is done in *table 4.11* for trial block 1, *table 4.12* for trial block 2, *table 4.13* for trial block 3 and *table 4.14* for trial block 4. The first column names the pattern, the second and forth the rank in the high similarity ranking, which is ordered now by the overall duration of the eye fixation. The third and fifth column give the similarity of the average vector of the first three places to the target pattern. The values in brackets are the already discussed values for the high similarity ranking ordered by the frequency. They are printed again to make it easier to compare both high similarity rankings.

Table 4.11: Target pattern rankings in high similarity list using duration as weight, block 1

Pattern	Ranking R1	Sim. AvgP. R1	Ranking R0	Sim. AvgP. R0
A1	Rank 2 (4)	100%(66%)	Rank 23 (1)	33%(88%)
B1	Rank 2 (2)	88%(100%)	Rank 25 (50)	66%(33%)
C1	Rank 103 (76)	44%(33%)	Rank 352 (239)	44%(55%)
D1	Rank 112 (2)	77%(88%)	Rank 6 (13)	55%(55%)
E1	Rank 63 (5)	55%(55%)	Rank 88 (1)	88%(100%)

Table 4.12: Target pattern rankings in high similarity list using duration as weight, block 2

Pattern	Ranking R1	Sim. AvgP. R1	Ranking R0	Sim. AvgP. R0
A2	Rank 86 (14)	55%(77%)	Rank 2 (301)	88%(55%)
B2	Rank 1005 (956)	22%(33%)	Rank 969 (1111)	22%(11%)
C2	Rank 1361 (1159)	44%(22%)	Rank 3192(2327)	11%(33%)
D2	Rank 6004 (4683)	44%(44%)	Rank 10265(10225)	44%(33%)
E2	Rank 1292 (176)	66%(22%)	Rank 1430 (1413)	11%(11%)

Including the duration as a sorting factor for the rankings showed no improvements. Also the rankings by duration seem to be overall worse than the rankings by frequency.

Table 4.13: Target pattern rankings in high similarity list using duration as weight, block 3

Pattern	Ranking R1	Sim. AvgP. R1	Ranking R0	Sim. AvgP. R0
A3	Rank 87 (12)	55%(66%)	Rank 32 (18)	66%(66%)
B3	Rank 59 (61)	44%(44%)	Rank 92(105)	33%(44%)
C3	Rank 69 (3)	77%(88%)	Rank 154(4)	77%(77%)
D3	Rank 42 (8)	33%(77%)	Rank 24 (1)	44%(100%)
E3	Rank 112 (6)	66%(66%)	Rank 91 (42)	77%(77%)

Table 4.14: Target pattern rankings in high similarity list using duration as weight, block 4

Pattern	Ranking R1	Sim. AvgP. R1	Ranking R0	Sim. AvgP. R0
A4	Rank 273 (52)	66%(66%)	Rank 161 (259)	44%(44%)
B4	Rank 2493 (457)	11%(77%)	Rank 357 (203)	66%(44%)
C4	Rank 3501 (5163)	33%(33%)	Rank 199 (899)	55%(55%)
D4	Rank 869 (886)	44%(55%)	Rank 3247 (1965)	55%(66%)
E4	Rank 651 (580)	66%(11%)	Rank 2684 (412)	55%(0%)

4.3.2 Discussion Experiment I

The comparison of the color distribution of all FPOIs for 2 colors with the color distribution of the target pattern shows a direct connection between both distributions - if white is the stronger color in the target pattern, the overall color distribution of all FPOIs for white is higher, and vice versa. For 3 colors it seemed that subjects where attracted in all trials by mostly black areas. The second color mostly depended on the stronger color in the target pattern color distribution. It is to mention that all trials had a white background, and the used gray color had a high contrast to this background. This means that dark colors had a higher contrast compared to white. A high contrast is (*following [9]*) more appealing to eye fixations, which is an explanation for this effect.

The analysis of the high similarity rankings for 2 colors showed very similar patterns to the target pattern in the first placements of the list. For 3 colors the results had no peculiar characteristics. The closer the FPOIs were to the target pattern, the better were also the high similarity rankings.

Regarding the results for the sub-fixations, it seems that the subjects always searched at least for a part of the pattern, in the case of 2 and 3 colors. Especially for 3 colors with 81 possible sub-fixation patterns, the rankings showed that a sub-fixation which was part of the target pattern was always on first or second place.

Combined with the poor results in the high similarity rankings, this could indicate that a pattern with 3 colors was to complex for a subject to look for. Therefore, the subject searched only for a part of the target pattern. After this part was found, the surrounding squares where compared with the whole target pattern.

Regarding the high similarity rankings ordered by duration there are indicators that the subject tended to look at patterns for a longer time which are similar to the sub-pattern that is in his mind, but the rest of the observed pattern was essentially different to the target pattern. It seems that subjects needed only a short time period to recognize patterns which are really similar (e.g., match in 7 or 8 out of 9 positions). If only a part is similar, but the rest is more subtly different, it takes longer for the subject to realize that there is a mismatch. E.g. this is the case if the pattern on the screen is upside-down or rotated. Therefore, the duration is an indicator for patterns that are somehow similar - either they are rotated, or have parts in it which are similar to the part of the pattern the subject is looking for - but it can not be said in which way this pattern is similar (e.g. that the observed pattern is upside-down). Hence, no additional information about the target pattern could be extracted out of the duration of a subject's fixation.

One big problem of experiment I was that the number of fixations and therefore the number of FPOIs was too low, so that its results could be impacted by

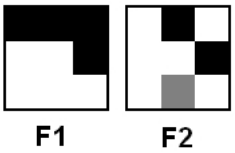


Figure 4.12: Patterns F1 and F2

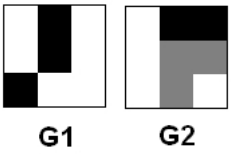


Figure 4.13: Patterns G1 and G2

too unspecific eye movement data. (E.g., the first 30 patterns in a high similarity ranking had all the same frequency, so the first 30 would have been all in first place.) Especially for 3 colors more eye movement data would have been needed. Also, the subjects could scan the search picture in a systematical way to find the target pattern which could influence the algorithm.

In order to avoid the systematical scanning and to receive more empirical data, a second experiment with a smaller square size was conducted. Because of the smaller square size the search picture was too big for subjects to scan it systematically. Furthermore, more time and more eye movements were needed to find the target pattern. With the second experiment the previous assumptions should be validated.

4.3.3 Reports Experiment II

The main difference between experiment I and II is the smaller square size. The square size is now 25x25 pixel, compared to 60x60 and 40x40 in experiment I. For experiment II the number of fixations were much higher than in experiment I.

Experiment II was run with 9 subjects. The two target patterns of block 1 and 2 for subject F are depicted in *figure 4.12*, for subject G in *figure 4.13*, for subject H in *figure 4.14*, for subject I in *figure 4.15*, for subject J in *figure 4.16*, for subject K in *figure 4.17*, for subject L in *figure 4.18*, for subject M in *figure 4.19* and for subject N in *figure 4.20*

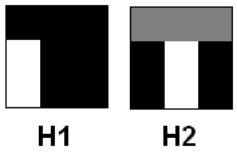


Figure 4.14: Patterns H1 and H2

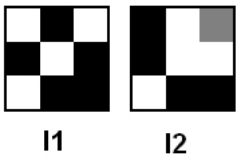


Figure 4.15: Patterns I1 and I2

Figure 4.16: Patterns J1 and J2

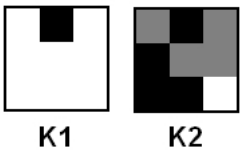


Figure 4.17: Patterns K1 and K2

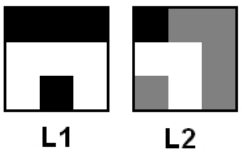


Figure 4.18: Patterns L1 and L2

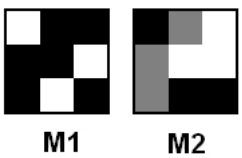


Figure 4.19: Patterns M1 and M2

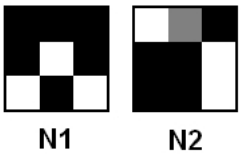


Figure 4.20: Patterns N1 and N2

Color Distribution

In the discussion for experiment I (*see section4.3.2*) it was assumed that depending on the target pattern color distribution also the color distribution of the FPOIs are influenced. *Table 4.15* shows the color distribution for block 1 of experiment II (2 colors), and *table 4.16* the color distribution for block 2 (3 colors).

Table 4.15: Color distribution block 1 (Exp. 2)

Pattern	Target Pattern Distr.	FPOI R1	FPOI R0
F1	56%/44%	50%/50%	50%/50%
G1	67%/33%	51%/49%	51%/49%
H1	22%/78%	41%/59%	38%/62%
I1	44%/56%	49%/51%	49%/51%
J1	44%/56%	51%/49%	52%/48%
K1	89%/11%	57%/43%	59%/41%
L1	56%/44%	54%/46%	56%/44%
M1	33%/67%	48%/52%	48%/52%
N1	33%/67%	47%/53%	47%/53%
Values are white/black percentages			

Table 4.16: Color distribution block 2 (Exp. 2)

Pattern	Target Pattern Distr.	FPOI R1	FPOI R0
F2	67%/22%/11%	38%/31%/31%	39%/30%/30%
G2	44%/22%/33%	36%/32%/31%	37%/32%/31%
H2	22%/44%/33%	30%/36%/33%	30%/37%/33%
I2	44%/44%/11%	32%/36%/31%	32%/37%/31%
J2	33%/44%/22%	31%/36%/33%	31%/36%/33%
K2	11%/44%/44%	30%/37%/32%	29%/38%/32%
L2	33%/11%/56%	40%/28%/31%	42%/26%/31%
M2	33%/33%/33%	34%/33%/32%	35%/32%/32%
N2	33%/56%/11%	30%/40%/30%	29%/42%/29%
Values are white/black/gray percentages			

For 2 colors, the color distribution of the target pattern does in fact influence the color distribution of the FPOIs. The bigger the proportion of one color in the pattern, the bigger it is in the color distribution.

For 3 colors the strongest color in the pattern was most of the times the strongest color in the color distribution (except for M2).
Regarding M2: Looking at the top sub-fixations in the sub-fixation rankings the subject probably looked for a part of the pattern which was mostly white. Therefore, the color distribution of these FPOIs are dominated by white.

Fixation and High Similarity Rankings

Table 4.17 shows the high similarity ranking for 2 colors which most of the times includes the target pattern in the top 3 ranks. Also, the average vector generated of the first three places shows a high similarity to the target pattern (mostly 7 or 8 out of 9 accordance).

Table 4.17: Target pattern rankings in high similarity list, block 1

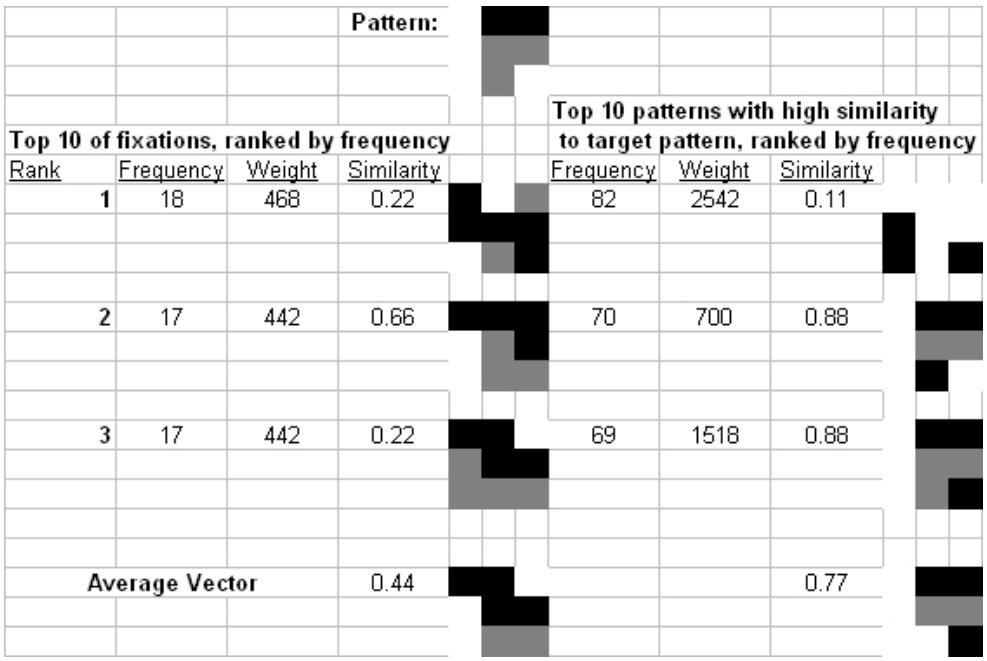
Pattern	Ranking R1	Sim. AvgP. R1	Ranking R0	Sim. AvgP. R0
F1	Rank 3	100%	Rank 12	66%
G1	Rank 1	100%	Rank 2	100%
H1	Rank 3	77%	Rank 2	77%
I1	Rank 2	88%	Rank 2	88%
J1	Rank 2	88%	Rank 1	88%
K1	Rank 1	88%	Rank 2	88%
L1	Rank 2	77%	Rank 2	77%
M1	Rank 13	77%	Rank 18	77%
N1	Rank 1	88%	Rank 1	77%

Table 4.18 disproves the same concept for 3 colors - the target pattern was most of the time on a very low rank.

Table 4.18: Target pattern rankings in high similarity list, block 2

Pattern	Ranking R1	Sim. AvgP. R1	Ranking R0	Sim. AvgP. R0
F2	Rank 180	88%	Rank 871	33%
G2	Rank 6879	77%	Rank 3584	33%
H2	Rank 837	44%	Rank 3010	33%
I2	Rank 952	33%	Rank 5079	33%
J2	Rank 9	55%	Rank 37	33%
K2	Rank 1052	44%	Rank 2015	44%
L2	Rank 694	55%	Rank 497	44%
M2	Rank 7	66%	Rank 205	33%
N2	Rank 999	55%	Rank 711	55%

But besides the poor results for 3 colors, in some cases the high similarity rankings show that the target pattern shape could be revealed - *see figure 4.21 and figure 4.22*. It seems that the high similarity ranking and its algorithm works well for 2 colors - but with an additional color the algorithm does not show the desired behavior. One reason could be the number of possible patterns for 3 colors - there are 19683 possible patterns, compared to 512 for 2 colors.



Pattern:						
				Top 10 patterns with high similarity		
Top 10 of fixations, ranked by frequency				to target pattern, ranked by frequency		
Rank	Frequency	Weight	Similarity	Frequency	Weight	Similarity
1	18	468	0.22	82	2542	0.11
2	17	442	0.66	70	700	0.88
3	17	442	0.22	69	1518	0.88
Average Vector			0.44			0.77

Figure 4.21: Pattern G2 fixation and high similarity ranking, radius 1

Sub-fixations

Table 4.19 shows that in trial block 1 the sub-fixation on first place in the sub-fixation ranking was most of the time also part of the target pattern. If not, rank 2 was it then. *Table 4.20* shows a similar result for trial block 2 with 3 colors - in most of the times the top sub-fixation on rank one or two was also part of the target pattern.

Duration of Fixations

The high similarity rankings ordered by duration are shown in *table 4.21* for 2 colors and in *table 4.22* for 3 colors. The high similarity rankings have become worse when ordered by duration instead by frequency.

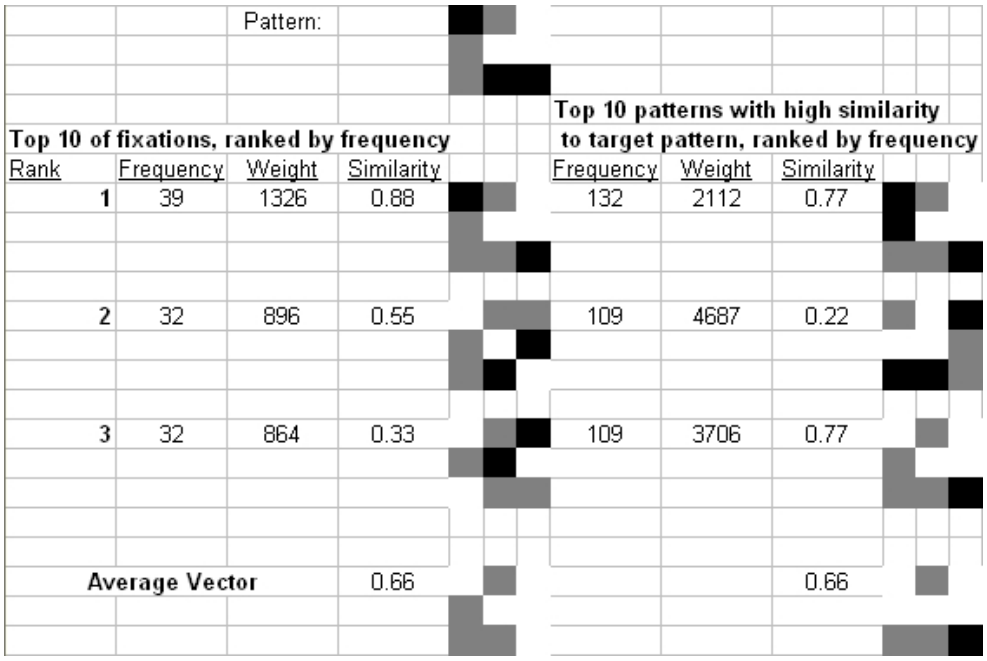


Figure 4.22: Pattern M2 fixation and high similarity ranking, radius 1

Table 4.19: First place in sub-fixation rankings is part of target pattern, block 1

Pattern	Block 1 R1	Block 1 R0
F1	No (Rank 2)	No (Rank 2)
G1	Yes	Yes
H1	Yes	Yes
I1	Yes	Yes
J1	Yes	Yes
K1	Yes	Yes
L1	No (Rank 2)	Yes
M1	No (Rank 2)	No (Rank 2)
N1	Yes	Yes

Table 4.20: First place in sub-fixation rankings is part of target pattern, block 2

Pattern	Block 2 R1	Block 2 R0
F2	Yes	Yes
G2	No (Rank 2)	No (Rank 2)
H2	No (Rank 10)	No (Rank 9)
I2	No (Rank 3)	Yes
J2	Yes	Yes
K2	Yes	No (Rank 2)
L2	No (Rank 3)	No (Rank 2)
M2	Yes	Yes
N2	Yes	Yes

Table 4.21: Target pattern rankings in high similarity list using duration as weight, block 1

Pattern	Ranking R1	Sim. AvgP. R1	Ranking R0	Sim. AvgP. R0
F1	Rank 213 (3)	66%(100%)	Rank 177 (12)	66%(66%)
G1	Rank 101 (1)	77%(100%)	Rank 44 (2)	33%(100%)
H1	Rank 29 (3)	33%(77%)	Rank 36 (2)	66(77%)
I1	Rank 2 (2)	88(88%)	Rank 27 (2)	55%(88%)
J1	Rank 175 (2)	11%(88%)	Rank 66 (1)	22%(88%)
K1	Rank 91 (1)	66%(88%)	Rank 15 (2)	66%(88%)
L1	Rank 1 (2)	88%(77%)	Rank 14 (2)	55%(77%)
M1	Rank 3 (13)	88%(77%)	Rank 12 (18)	44%(77%)
N2	Rank 111(1)	88%(88%)	Rank 37 (1)	22(77%)

Table 4.22: Target pattern rankings in high similarity list using duration as weight, block 2

Pattern	Ranking R1	Sim. AvgP. R1	Ranking R0	Sim. AvgP. R0
F2	Rank 2391 (180)	55%(88%)	Rank 910 (871)	55%(33%)
G2	Rank 6565 (6879)	66%(77%)	Rank 3347 (3584)	33%(33%)
H2	Rank 1037 (837)	44%(44%)	Rank 3383 (3010)	55%(33%)
I2	Rank 9656 (952)	44%(33%)	Rank 5577 (5079)	33%(33%)
J2	Rank 575 (9)	55%(55%)	Rank 247 (37)	33%(33%)
K2	Rank 5854 (1052)	33%(44%)	Rank 6442 (2015)	33%(44%)
L2	Rank 643 (694)	33%(55%)	Rank 2471 (497)	44%(44%)
M2	Rank 326 (7)	11%(66%)	Rank 593 (205)	66%(33%)
N2	Rank 113 (999)	55%(55%)	Rank 207 (711)	66%(55%)

4.3.4 Discussion Experiment II

For 2 colors, the color distribution of the target pattern does in fact influence the color distribution of the FPOIs. The bigger the proportion of one color in the target pattern, the bigger it is in the color distribution of all FPOIs. For 3 colors the strongest color in the target pattern was most of the times the strongest color in the color distribution of the FPOIs.

It seems that the color distribution of the FPOIs and therefore the subjects' fixations are mainly, but not only influenced by the target pattern. As already mentioned in *section 4.3.2* especially for 3 colors a high contrast is (*following [9]*) more appealing to eye fixations, so that black and gray could not really be differentiated by the subject.

The assumption that a subject looks at similar patterns when searching for the target pattern can be verified for 2 colors - the high similarity rankings for 2 colors show that the algorithm to create these rankings works well and that its produced patterns are similar to the target pattern. For 3 colors the algorithm is working poorly. It seems that a target pattern with 3 colors is too complex for the human eye to search for it in a holistic way - eye movements and the fixations are becoming more "uncertain".

The assumption that the duration of a fixation does not provide any additional information about the similarity to the target pattern can be verified - if the high similarity list is ranked by the subject's duration, the rankings become worse. There are no additional information in the duration of a subject's fixation about the similarity to the target pattern.

The ranking for the sub-fixations proofed the idea that subjects search primarily for a part of the target pattern. In the rankings, a sub-fixation which is part of the target pattern is almost always in first or second place. Therefore, the subject first searches for the sub-pattern; after he found a sub-pattern, he compares the surroundings squares with the rest of the target pattern.

Furthermore, reducing the square size avoided the effect that subject's could scan the search picture systematically. Also, more fixations and therefore more empirical data were generated.

Chapter 5

Conclusion

In this work an introduction into the field of eye tracking was given. A very common task in eye tracking is to examine search behavior of subjects in different search tasks - a lot of work has already been done in this area. Especially it was found out that if a basic shape were given to subjects and they had to find this in a "random noise stimuli" (where the noise matched the average spectrum of natural images) the eye movements tend to fixate areas that were similar to the stimuli. An experiment was introduced were a target pattern and a stimuli consisting of squares in either two or three colors with different square sizes were presented to the subject. The subjects were to find this target pattern in the search picture. The question was if it is possible to calculate the target pattern just with the given eye movements, without any knowledge of the target pattern itself. (*See chapter 2*)

The software developed and used to create and run the experiment and to analyze the output data was introduced and described in detail. The three software parts named Experiment Builder, Experiment Engine and Experiment Analysis Tool are the content of *chapter 3*.

The terms **ROI** and **FPOI** were explained. The ROI (*region of interest*) is the area of surrounding squares of a subject's fixation in the search picture. The size of the ROI depends on a radius parameter. The FPOI (*fixation pattern of interest*) is a pattern extracted from the ROI with the size of the target pattern. Furthermore, the algorithms and parameters to create different rankings with the Experiment Analysis Tools were described in detail.

Chapter 4 conducts an experiment with 14 subjects. The experiment was split up into two different experiment setups and different block of trials for each setup. The first experiment was conducted with 5 and the second experiment with 9 participants. The main difference between the two experiments was the smaller square size and because of this a higher number of fixations in the sec-

ond experiment. Both experiments contained search task with two and three colors. The experiment results and reports created with the Experiment Analysis tool were summarized and discussed.

The results compare the color distribution of the target pattern with the overall color distrubtion of all FPOIs. Regarding the results of the experiments it can be said that the color distribution of the target pattern influences the fixations of the subject. Also the stronger color in the target pattern is the stronger color in the color distribution of all FPOIs. For 3 colors more factors (like the contrast of the colors to the background) are involved, but the results show that the strongest color in the target pattern is likely to be the strongest color in the color distribution of all FPOIs.

A list of patterns that are very similar to each FPOI (8 out of 9 positions in the pattern have the same color as the FPOI) is built. This list was ordered by the amount of occurrences (frequency) of each pattern. For the experiments with 2 colors, the list contained more often the target pattern as other patterns. For 3 colors, the basic shape could be revealed in some results, but no special correlation between the first places in the ranking and the target pattern could be observed.

Creating the same ranking of patterns with high similarity, but ordering it by the overall duration of the subjects' fixation instead by frequency, the rank of the target pattern in this ranking got worse. The duration of a subject fixation did not contain additional information for the algorithm. That means that patterns which the subject fixated for a longer time are not more similar to the target pattern than patterns which the subject fixated for a shorter time period.

A list of all sub-fixations (a list of all 2x2 patterns inside a 3x3 FPOI) was created and ranked by their frequency. Surprisingly, the results showed that the sub-fixations on first places were in almost all experiments part of the target pattern. Therefore, the subject memorized during his search task only a part of the target pattern and searched for this part in the search pictures. When she/he found this part in the picture, he/she then compared the surrounding squares with the target pattern, and continued or finished searching.

Reducing the square size of the target pattern and search picture led to "better" results - by reducing the square size, a systematical scanning of the stimuli by a subject was avoided. Furthermore, more fixations and therefore more empirical data could be gathered, which improved the results of the algorithm.

Regarding the results for 3 colors, and compared to the results with 2 colors, it can be concluded that 3 colors are too complex for a subject to search in a holistic way. This means that the subject is not able to search for the pattern "in a whole" - he/she only looks for a part of the pattern, and then has to compare the surroundings with the target. For 2 colors the subject probably can look for the target pattern "in a whole", because it is less complex, so he/she can fixate

easier areas which are more similar.

For 2 colors the algorithm does produce good results, and also a smaller square size led to improvements. Therefore, it can be assumed that the algorithm works well with simple patterns that are black and white (or consist of two colors with a high contrast) and that the calculated pattern is very similar (or the same) to the target pattern. Furthermore, a smaller square size is advantageous for the algorithm.

An application for this algorithm could be in image databases and content-based image retrieval. Supposing a subject is very sensitive to certain patterns or colors, the next time when she/he searches an image database, the algorithm could be used to list the images which match to his search purposes.

Another application for the algorithm could be in the field of human-computer interaction: there it is important to know the user's intention. It could help the user when he/she has problems finding a specific button or menu item. The algorithm could help to generate suggestions that comply with his search purpose.

Bibliography

[1] B.L. Beard and A.J. Ahumada. A technique to extract relevant image features for visual tasks. *Human Vision and Electronic Imaging III. SPIE Proceedings*, 3299:79–85, 1998. Bellingham, WA: SPIE (In B. E. Rogowitz & T. N. Pappas). [cited at p. 10]

[2] D.C. Burr, M.C. Morrone, and J. Ross. Selective suppression of the magnocellular visual pathway during saccadic eye movements. *Nature*, 371:511Y513, 1994. [cited at p. 6]

[3] SDL Community. Sdl (simple direct medialayer) website. http://www.libsdl.org. [cited at p. 28]

[4] B. Efron. *An introduction to the bootstrap.* Boca Raton, FL: Chapman & Hall/CRC., 1994. [cited at p. 11]

[5] J.M. Findlay. Saccade target selection during visual search. vision research. *Vision Research*, 37:617Y631, 1997. [cited at p. 8]

[6] E. Hess and J. Polt. Pupil size as related to interest value of visual stimuli. *Science*, 140, 1960. [cited at p. 37]

[7] M.A. Just and P.A. Carpenter. The psychology of reading and language comprehension. *Boston: Allyn & Bacon*, 1987. [cited at p. 5]

[8] Universitt Konstanz. knime - konstanz information miner. http://www.knime.org. [cited at p. 42]

[9] L.Itti and C. Koch. A saliency-based search mechanism for overt and covert shifts of visual attention. *Vision Research*, 40(10-12):1489–1506, 2000. [cited at p. 64, 73]

[10] Microsoft. Microsoft download. http://www.microsoft.com/downloads/. [cited at p. 29, 42]

[11] D.J. Parkhurst and E. Niebur. Scene content selected by active vision. *Spatial Vision*, 16:125–154, 2003. [cited at p. 9]

[12] M. Pomplun. *Analysis and Models of Eye Movements in Comparative Visual Search.* PhD thesis, University Bielefeld, 1996. [cited at p. 6]

[13] M. Pomplun. Saccadic selectivity in complex visual search displays. *Vision Research*, 46:1886–1900, 2006. [cited at p. 9, 12]

[14] U. Rajashekar, A. Bovik, and L. Cormack. Visual search in noise: Revealing the influence of structural cues by gaze-contingent classification image analysis. *Journal of Vision*, 6:379–386, 2006. [cited at p. 9, 10, 11, 12, 150]

[15] P. Reinagel and A.M. Zador. Natural scene statistics at the centre of gaze. *Network*, 10:341–350, 1999. [cited at p. 9]

[16] SR Research. Eye tracker ii overview. http://www.sr-research.com/mount_main.php. [cited at p. 7]

[17] SR Research. *EyeLink Programmers Guide*, version 3.0 edition. [cited at p. 28]

[18] SR Research. Sr research homepage. http://www.srresearch.com. [cited at p. 28]

[19] SR Research. *EyeLink II User Manual*, version 2.11 edition, 1997-2005. [cited at p. 26]

[20] R.F. Murray R.F and (et Al). Saccadic and perceptual performance in visual search tasks: Ii. letter discrimination. *Journal of the Optical Society of America A, Optics, Image Science and Vision*, 20:1356–1370, 2003. [cited at p. 8]

[21] Wikipedia. Eye tracking. http://en.wikipedia.org/wiki/Eye_tracking, 07 2007. [cited at p. 6]

[22] L.G. Williams. The effects of target specification on objects fixated during visual search. *Acta Psychologica*, 27:355Y360, 1967. [cited at p. 8]

[23] J.M. Wolfe and T.S. Horowitz. What attributes guide the deployment of visual attention and how do they do it? *Neuroscience*, 5(495Y501), 2004. [cited at p. 8]

[24] A.L. Yarbus. Eye movements and vision. *New York Plenum Press*, 1967. [cited at p. 5]

Appendices

Appendix A

Used software and CD contents

A.1 Used Software

The following tools were used to develop and implement the code:

- *Visual Studio 2005 Service Pack 2 from Microsoft*
  An IDE for developing C++ & .NET-Application. It includes *Microsoft .NET 2.0*.
- *Microsoft Office Primary Interop Assemblies (PIA) for Office*
  API for *Microsoft Office*, including Excel.
- *Visual Paradigm for UML 6.1 Community Edition*
  An UML-Tool that integrates into Visual Studio and supports creating UML diagrams.
- *nDoc Beta 2005*
  An open source tool to create Documentation files from XML-documented .NET-Sourcecode.
- *Ghostdoc*
  A freeware Visual Studio plug-in to generate automated code documentation.
- *kNime*
  Konstanz Information Miner, a data-mining tool.
- *SR Research EyeTracker II Software Library*
  Tools and templates useful for developing EyeLink II experiments.

This work was created with *Latex*, using *MikTex 2.6* and *LEd (LatexEditor) Version 0.51*. Snapshots were made with *MWSwap 3*.

A.2 Contents of the Enclosed CD

- `ASCLibrary`
  This folder contains the *ASCLibrary* project files and source codes, as described as part of the *Experiment Analysis Tool* in section 3.3.3.

- `BMPAnalysis`
  This folder contains the *Experiment Analysis Tool* project files and sources codes, as described in section 3.3.3.

- `BMPBuilder`
  This folder contains the *Experiment Builder* project files and source codes, as described in section 3.1.3. Furthermore, the solution file that contains the Experiment Analysis Tool, Experiment Builder and Experiment Engine projects is inside this folder.

- `BMPExperiment`
  This folder contains the *Experiment Engine* project files and sources codes, as described in section 3.2.2.

- `BMPExperimentData`
  The experiment created with the Experiment Builder, the eye movement data from the EyeTracker device generated during the experiment runs as well as the corresponding reports generated by the Experiment Analysis Tool are (separated for each subject) inside this folder.

- `Docs`
  The code documentation.

Appendix B

Namespace ASCLibrary

*N*amespace Contents Page

Classes

B.1 Classes

B.1.1 Class ASCBlink

Contains all information about a blink event in an ASC-Data File.

Declaration

```
public class ASCBlink
    : Object
```

Properties

- *D*uration

  public **i**nt **D**uration { get; set; }

  Gets or sets the duration of the blink in ms.

- *E*ndTime

  public **i**nt **E**ndTime { get; set; }

  Gets or sets the end time of the blink in ms, relative to the start of the experiment.

- *S*tartTime

  public **i**nt **S**tartTime { get; set; }

  Gets or sets the start time of the blink in ms, relative to the start of the experiment.

Constructors

- .ctor

  public **A**SCBlink( )

B.1.2 Class ASCData

Contains all information of events in an ASC-Data File, that is extracted from an .edf-File of an EyeTracker experiment.

Declaration

public class ASCData
: Object

Properties

- *T*rialDataList

  public **S**ystem.Collections.Generic.List{ASCLibrary.ASCTrialData} **T**rialDataList { get; set; }

  Gets or sets the trial data list.

Constructors

- .ctor

  public **A**SCData( )

Methods

- *L*oadData

  public **v**oid **L**oadData( )

  Loads the data of an ASC-File.

  – **P**arameters

  * filename - The filename.

B.1.3 Class ASCFixation

Contains all information about a fixation event in an ASC-Data File.

Declaration

```
public class ASCFixation
    : Object
```

Properties

- *A*veragePupilSize

  public **i**nt **A**veragePupilSize { get; set; }

  Gets or sets the average size of the pupil in pixel.

- *A*verageX

  public **i**nt **A**verageX { get; set; }

  Gets or sets the average X fixation value on the screen.

- *A*verageY

  public **i**nt **A**verageY { get; set; }

  Gets or sets the average Y fixation value on the screen.

- *D*uration

  public **i**nt **D**uration { get; set; }

  Gets or sets the duration of the fixation in ms.

- *E*ndTime

  public **i**nt **E**ndTime { get; set; }

  Gets or sets the end time of the fixation in ms, relative to the start of the experiment.

- *P*osX

  public **i**nt **P**osX { get; set; }

  Gets or sets the X-Position of the fixation. Not used/deprecated.

- *P*osY

  public **i**nt **P**osY { get; set; }

  Gets or sets the Y-Pos of the fixation. Not used/deprecated

- *S*tartTime

  public **i**nt **S**tartTime { get; set; }

  Gets or sets the start time of the fixation in ms, relative to the start of the experiment.

Constructors

- .ctor

  public **A**SCFixation( )

B.1.4 Class ASCSaccation

Contains all information about a saccade event in an ASC-Data File.

Declaration

```
public class ASCSaccation
    : Object
```

Properties

- *D*uration

  public **i**nt **D**uration { get; set; }

  Gets or sets the duration in ms.

- *E*ndTime

  public **i**nt **E**ndTime { get; set; }

  Gets or sets the end time in ms, relative to the start of the experiment.

- *E*ndX

  public **i**nt **E**ndX { get; set; }

  Gets or sets the end X-position on the screen.

- *E*ndY

  public **i**nt **E**ndY { get; set; }

  Gets or sets the end Y-position on the screen.

- *P*eakVelocity

  public **i**nt **P**eakVelocity { get; set; }

  Gets or sets the peak velocity.

- *S*accadicAmplitude

  public **d**ecimal **S**accadicAmplitude { get; set; }
  Gets or sets the saccadic amplitude.

- *S*tartTime

  public **i**nt **S**tartTime { get; set; }
  Gets or sets the start time in ms, relative to the start of the experiment.

- *S*tartX

  public **i**nt **S**tartX { get; set; }
  Gets or sets the start X-position on the screen.

- *S*tartY

  public **i**nt **S**tartY { get; set; }
  Gets or sets the start Y-position on the screen.

Constructors

- .ctor

  public **A**SCSaccation( )

B.1.5 Class ASCTrialData

Contains the event information of a trial, namely all saccades, fixations, blinks and the trial metadata.

Declaration

```
public class ASCTrialData
    : Object
```

Properties

- *B*linkList

  public **S**ystem.Collections.Generic.List{ASCLibrary.ASCBlink} **B**linkList { get; set; }

  Gets or sets the blink list.

- *F*ixationList

  public **S**ystem.Collections.Generic.List{ASCLibrary.ASCFixation} **F**ixationList { get; set; }

  Gets or sets the fixation list.

- *S*accationList

  public **S**ystem.Collections.Generic.List{ASCLibrary.ASCSaccation} **S**accationList { get; set; }

  Gets or sets the saccation list.

- *T*rialMetaData

  public **A**SCLibrary.ASCTrialMetaData **T**rialMetaData { get; set; }

  Gets or sets the trial meta data.

Constructors

- .ctor

  public **A**SCTrialData( )

B.1.6 Class ASCTrialMetaData

Contains all metadata information of an ASC-File

Declaration

```
public class ASCTrialMetaData
    : Object
```

Properties

- *E*ndTime

  public **i**nt **E**ndTime { get; set; }

  Gets or sets the end time.

- *S*tartTime

  public **i**nt **S**tartTime { get; set; }

  Gets or sets the start time.

- *T*imeout

  public **b**ool **T**imeout { get; set; }

  Gets or sets a value indicating whether this trial timed out.

- *T*imeoutTime

  public **i**nt **T**imeoutTime { get; set; }

  Gets or sets the time after which the time out happened.

- *T*rialID

  public **i**nt **T**rialID { get; set; }

  Gets or sets the trial ID.

- *T*rialName

  public **s**tring **T**rialName { get; set; }

  Gets or sets the name of the trial.

Constructors

- .ctor

  public **A**SCTrialMetaData( )

Appendix C

Namespace BMPAnalysis

*N*amespace Contents Page

Classes

Creates and display text files that contain raw data with the fixations and results of the analysis.

Vector 115

A vector is a one-dimensional representation of a two dimensional pattern. If the pattern is e.g. a 3x3 array of different color values, a vector of this pattern would be an one-dimensional array with the length 9. The first three color values of the vector array would be identical with the top three color values of the pattern array.(The top left, top center and top right colors). The second group of three would correspond to the colors in the middle row, and the the last three values would be the ones in the bottom row. Therefore, the conversion is row wise from top to bottom. A vector can be written as a string by writing the abbreviated color letters in a single row (E.g.: ”WWWB” is a vector string for describing a vector with the color sequence white,white,white and black.)

C.1 Classes

C.1.1 Class AnalysisObject

This object contains all data needed to generate an analysis.

Declaration

```
public class AnalysisObject
    : Object
```

Properties

- *A*scData

  public **A**SCLibrary.ASCData **A**scData { get; set; }
  Gets or sets the asc data.

- *C*onfig

  public **B**MPAnalysis.Config **C**onfig { get; set; }
  Gets or sets the config data.

- *D*ataFile

  public **s**tring **D**ataFile { get; set; }
  Gets or sets the data file.

- *E*xperiment

  public **B**MPBuilder.ExperimentData **E**xperiment { get; set; }
  Gets or sets the experiment data.

- *E*xperimentFolder

  public **s**tring **E**xperimentFolder { get; set; }
  Gets or sets the experiment folder.

- *N*ame

  public **s**tring **N**ame { get; set; }

  Gets or sets the name.

- *P*arameterList

  public **B**MPAnalysis.OutputFileParameter **P**arameterList { get; set; }

  Gets or sets the parameter list.

- *P*atternList

  public **S**ystem.Collections.Generic.List{BMPBuilder.Pattern} **P**atternList { get; set; }

  Gets or sets the pattern list.

- *P*ictureList

  public **S**ystem.Collections.Generic.List{BMPBuilder.Picture} **P**ictureList { get; set; }

  Gets or sets the picture list.

Constructors

- .ctor

  public **A**nalysisObject( )

C.1.2 Class ColorDistribution

Defines a color distribution of two or more colors - a distribution contains the percentage.

Declaration

```
public class ColorDistribution
    : Object
```

Properties

- *F*requency

  public **l**ong **F**requency { get; set; }

  Gets or sets the frequency.

- *P*ercentage

  public **d**ouble **P**ercentage { get; set; }

  Gets or sets the percentage.

Constructors

- .ctor

  public **C**olorDistribution( )

  Initializes a new instance of the class.

  – **P**arameters

  * percentage - The percentage.
  * frequency - The frequency.

Methods

- CompareTo

  public **i**nt **C**ompareTo( )

  Compares the current instance with another object of the same type.

  – **P**arameters

    * obj - An object to compare with this instance.

C.1.3 Class Config

Represents a configuration file which contains the resolution of the display screen and the overall number of trials of an experiment

Declaration

```
public class Config
    : Object
```

Properties

- *D*isplayX

  public **i**nt **D**isplayX { get; set; }

  Gets or sets the width of the display.

- *D*isplayY

  public **i**nt **D**isplayY { get; set; }

  Gets or sets the height of the display.

- *N*umTrials

  public **i**nt **N**umTrials { get; set; }

  Gets or sets the overall number of trials.

Constructors

- .ctor

  public **C**onfig( )

  Initializes a new instance of the class.

- .ctor

public **C**onfig( )

Initializes a new instance of the class.

- **P**arameters
  - filename - The filename of the config file to be load.

Methods

- *L*oad

public **v**oid **L**oad( )

Loads the specified config filef.

- **P**arameters
  - filename - The filename of the config file.

C.1.4 Class Info

Displays the Info window with description about the author and program.

Declaration

```
public class Info
    : Form
```

Constructors

- .ctor

  public **I**nfo( )

C.1.5 Class MainForm

Displays the main GUI for the analysis tool.

Declaration

```
public class MainForm
    : Form
```

Constructors

- .ctor

  public **M**ainForm( )

  Initializes a new instance of the class.

C.1.6 Class OutputFile

Analyzes the data and makes all the necessary calculations to create the analysis output file - to implement this class, the abstract methods should generate the equivalent output for its specified file format.

Declaration

public class OutputFile
: Object

Properties

- *A*scData

  public **A**SCLibrary.ASCData **A**scData { get; set; }
  Gets or sets the ASC data.

- *C*onfig

  public **B**MPAnalysis.Config **C**onfig { get; set; }
  Gets or sets the config data.

- *D*ataFile

  public **s**tring **D**ataFile { get; set; }
  Gets or sets the data file.

- *E*xperiment

  public **B**MPBuilder.ExperimentData **E**xperiment { get; set; }
  Gets or sets the experiment.

- *E*xperimentFolder

  public **s**tring **E**xperimentFolder { get; set; }
  Gets or sets the experiment folder.

- *P*arameter

  public **B**MPAnalysis.OutputFileParameter **P**arameter { get; set; }

  Gets or sets the parameter.

- *P*atternList

  public **S**ystem.Collections.Generic.List{BMPBuilder.Pattern} **P**atternList { get; set; }

  Gets or sets the pattern list.

- *P*atternVector

  public **B**MPAnalysis.Vector **P**atternVector { get; set; }

  Gets or sets the pattern vector.

- *P*ictureList

  public **S**ystem.Collections.Generic.List{BMPBuilder.Picture} **P**ictureList { get; set; }

  Gets or sets the picture list.

Constructors

- .ctor

  public **O**utputFile( )

  Initializes a new instance of the class.

  - **P**arameters
    - parameterList - The parameter list.
    - experiment - The experiment data.
    - pictureList - The picture list.
    - patternList - The pattern list.
    - config - The config data.
    - ascData - The ASC data.
    - dataFile - The data file.
    - experimentFolder - The experiment folder.

Methods

- CreateOutputFile

  `p`ublic **v**`oid` **C**reateOutputFile( )

  Creates the output file.

  – **P**arameters

    * `w`orker - The background worker object, to which progress is reported.

C.1.7 Class OutputFileExcel

Creates and displays excel sheets with the data output.

Declaration

```
public class OutputFileExcel
    : OutputFile
```

Constructors

- .ctor

  public **O**utputFileExcel( )

  Initializes a new instance of the class.

  – **P**arameters

    * parameterList - The parameter list.
    * experiment - The experiment data.
    * pictureList - The picture list.
    * patternList - The pattern list.
    * config - The config data.
    * ascData - The ASC data.
    * dataFile - The data file.
    * experimentFolder - The experiment folder.

C.1.8 Class OutputFileParameter

Represents the parameter for the calculation of the analysis and creating the output files.

Declaration

```
public class OutputFileParameter
    : Object
```

Properties

- *E*xcel

  public **b**ool **E**xcel { get; set; }

  Gets or sets a value indicating whether the output should be an Excel sheet.

  - **U**sage
    - Defines a XOR-Relationship between the excel and the rawData Flag.

- *F*rom

  public **i**nt **F**rom { get; set; }

  Gets or sets at which trial to start the output.

- *I*ncludeFirstFixation

  public **b**ool **I**ncludeFirstFixation { get; set; }

  Gets or sets a value indicating whether the first fixation of a trial should be included.

- *I*ncludePatternFixation

  public **b**ool **I**ncludePatternFixation { get; set; }

  Gets or sets a value indicating whether the pattern fixation should be included or not.

- *M*oveX

  public **i**nt **M**oveX { get; set; }

  Gets or sets if and where the fixation position should be moved in vertical direction. For instance, a value of -1 would move the fixation 1 square to the left.

- *M*oveY

  public **i**nt **M**oveY { get; set; }

  Gets or sets if and where the fixation position should be moved in horizontal direction. For instance, a value of -1 would move the fixation 1 square up.

- *O*utputFolder

  public **s**tring **O**utputFolder { get; set; }

  Gets or sets the output folder. Only signification if raw data (txt) files are desired.

- *R*adius

  public **i**nt **R**adius { get; set; }

  Gets or sets the radius. Indicates how many surrounding squares of a fixation (additional to the pattern size) should be included to the fixation calculation.

- *R*awData

  public **b**ool **R**awData { get; set; }

  Gets or sets a value indicating whether the output should be raw data (txt) files.

  - **U**sage
    - Defines a XOR-Relationship between the excel and the rawData Flag.

- *S*hortExcelReport

public **b**ool **S**hortExcelReport { get; set; }

Gets or sets a value indicating whether a short excel report should be generated.

- *S*imilarityMeasure

  public **d**ouble **S**imilarityMeasure { get; set; }

  Gets or sets the similarity measure, which is a percentage value that describes how similar two vectors are to each other.

- *T*o

  public **i**nt **T**o { get; set; }

  Gets or sets at which trial to end the output.

- *T*op10

  public **i**nt **T**op10 { get; set; }

  Gets or sets how many vectors should be included in the final ranking list.

Constructors

- .ctor

  public **O**utputFileParameter( )

Methods

- *T*oString

  public **s**tring **T**oString( )

  Returns a String that represents the current OutputFileParameter.

C.1.9 Class OutputFileRawData

Creates and display text files that contain raw data with the fixations and results of the analysis.

Declaration

```
public class OutputFileRawData
    : OutputFile
```

Constructors

- .ctor

  public **O**utputFileRawData( )

  Initializes a new instance of the class.

  – **P**arameters

  * parameterList - The parameter list.
  * experiment - The experiment data.
  * pictureList - The picture list.
  * patternList - The pattern list.
  * config - The config data.
  * ascData - The ASC data.
  * dataFile - The data file.
  * experimentFolder - The experiment folder.

C.1.10 Class Vector

A vector is a one-dimensional representation of a two dimensional pattern. If the pattern is e.g. a 3x3 array of different color values, a vector of this pattern would be an one-dimensional array with the length 9. The first three color values of the vector array would be identical with the top three color values of the pattern array.(The top left, top center and top right colors). The second group of three would correspond to the colors in the middle row, and the the last three values would be the ones in the bottom row. Therefore, the conversion is row wise from top to bottom. A vector can be written as a string by writing the abbreviated color letters in a single row (E.g.: "WWWB" is a vector string for describing a vector with the color sequence white,white,white and black.)

Declaration

public class Vector
: Object

Properties

- ColorList

  public **S**ystem.Collections.Generic.List{System.Drawing.Color} **C**olorList { get; set; }

  Gets or sets a list of all colors the vector can (but may not) contain.

- Colors

  public **S**ystem.Int32[] **C**olors { get; }

  Gets the color values as integers of the vector array.

- *F*requency

  public **l**ong **F**requency { get; set; }

  Gets or sets the frequency. If a number of vectors are combined in a list, the frequency describes the occurrences in this list.

- *S*ize

public **i**nt **S**ize { get; set; }

Gets or sets the size of the vector. This is basically the length of the vector array.

- *W*eight

  public **l**ong **W**eight { get; set; }

  Gets or sets the weight of the vector. This can be the duration of a fixation.

Constructors

- .ctor

  public **V**ector( )

  Initializes a new instance of the class.

  – **P**arameters
    * size - The size of the vector. This is basically the length of the vector array.
    * colorList - A list of all colors the vector can (but may not) contain.

- .ctor

  public **V**ector( )

  – **P**arameters
    * colorArray -
    * colorList -

- .ctor

  public **V**ector( )

  Initializes a new instance of the class.

  – **P**arameters
    * size - The size of the vector. This is basically the length of the vector array.
    * weight - The weight of the vector, e.g. the duration of a fixation.

    * colorList - A list of all colors the vector can (but may not) contain.

- .ctor

  public **V**ector( )

  Initializes a new instance of the class.

  – **P**arameters

    * vector - An existing vector that values will be copied into the new vector object, except its frequency and its weight.
    * weight - The new weight of the new vector.
    * frequency - The new frequency of the new vector.

Methods

- *A*dd

  public **v**oid **A**dd( )

  Adds the specified color to the vector. If the vector is newly instantiated, this will start at the first element of the vector, and continues with the next element, until the vector is filled.

  – **P**arameters

    * color - The color value as integer that should be added at the next unset vector position.

- *A*dd

  public **v**oid **A**dd( )

  Adds the specified color to the vector. If the vector is newly instantiated, this will start at the first element of the vector, and continues with the next element, until the vector is filled.

  – **P**arameters

    * color - The color value as color object that should be added at the next unset vector position.

- *A*verageVector

  public **B**MPAnalysis.Vector **A**verageVector( )

  Calculates a new average vector of a list of vectors. Counts the frequency (or the total weight) of each color for each position in all vectors, and writes the color with the highest frequency (or highest weight) in the new average vector. If the frequency for a position is the same for all colors, the color which has color the highest value in the color distribution is taken.

  – **P**arameters

    * vectorList - The vector list to build the new average vector of.
    * range - The range, how many of the vectors in the vector list should be considered. If the value is smaller than the size of the vector list, the first vectors up to the given range of the list will be taken
    * colorList - The list of all possible colors as an integer array that can, but may not occur in the vectors.
    * colorDistribution - The color distribution of all possible colors that can, but may not occur in the vectors.
    * useWeight - if set to true, the weight instead the frequency is used to build the average vector.

- *A*verageVector

  public **B**MPAnalysis.Vector **A**verageVector( )

  Calculates a new average vector of a list of vectors. Counts the frequency (or the total weight) of each color for each position in all vectors, and writes the color with the highest frequency (or highest weight) in the new average vector. If the frequency for a position is the same for all colors, the color which has color the highest value in the color distribution is taken.

  – **P**arameters

    * vectorList - The vector list to build the new average vector of.
    * colorList - The list of all possible colors as a list of color objects that can, but may not occur in the vectors.
    * colorDistribution - The color distribution of all possible colors that can, but may not occur in the vectors.
    * useWeight - if set to true, the weight instead the frequency is used to build the average vector.

- *A*verageVector

  public **B**MPAnalysis.Vector **A**verageVector( )

  Calculates a new average vector of a list of vectors. Counts the frequency (or the total weight) of each color for each position in all vectors, and writes the color with the highest frequency (or highest weight) in the new average vector. If the frequency for a position is the same for all colors, the color which has color the highest value in the color distribution is taken.

  – **P**arameters
    * vectorList - The vector list to build the new average vector of.
    * colorList - The list of all possible colors as an integer array that can, but may not occur in the vectors.
    * colorDistribution - The color distribution of all possible colors that can, but may not occur in the vectors.
    * useWeight - if set to true, the weight instead the frequency is used to build the average vector.

- *A*verageVector

  public **B**MPAnalysis.Vector **A**verageVector( )

  Calculates a new average vector of a list of vectors. Counts the frequency (or the total weight) of each color for each position in all vectors, and writes the color with the highest frequency (or highest weight) in the new average vector. If the frequency for a position is the same for all colors, the color which has color the highest value in the color distribution is taken.

  – **P**arameters
    * vectorList - The vector list to build the new average vector of.
    * range - The range, how many of the vectors in the vector list should be considered. If the value is smaller than the size of the vector list, the first vectors up to the given range of the list will be taken
    * colorList - The list of all possible colors as a list of color objects that can, but may not occur in the vectors.
    * colorDistribution - The color distribution of all possible colors that can, but may not occur in the vectors.
    * useWeight - if set to true, the weight instead the frequency is used to build the average vector.

- ColorDistribution

  public **S**ystem.Double[] **C**olorDistribution( )

  Calculates the percentage of the color distribution.

  - **P**arameters
    - decimals - How many decimal places the percentage should have.

- ColorDistributionString

  public **s**tring **C**olorDistributionString( )

  Returns a string which describes the percentage of the color distrubution.

  - **P**arameters
    - decimals - How many decimal places the percentage should have.

- Compare

  public **d**ouble **C**ompare( )

  Calculates the similarity of the current instance with another vector. The similarity is a percentage which indicates how many positions have the same value in both vectors.

  - **P**arameters
    - v1 - The vector to compare the current instance with.

- Compare

  public **d**ouble **C**ompare( )

  Calculates the similarity of two vectors with each other. The similarity is a percentage which indicates how many positions have the same value in both vectors.

  - **P**arameters
    - v1 - The first vector to compare with the second vector.
    - v2 -

- CompareTo

  public **i**nt **C**ompareTo( )

  Compares the current instance with another vector object. Uses the frequency to compare these vectors.

  - **P**arameters
    - obj - An object to compare with this instance.

- CompareToUsingWeight

  public **i**nt **C**ompareToUsingWeight( )

  Compares two vectors with each other. Uses the weight to compare these vectors.

  - **P**arameters
    - obj1 - The first vector
    - obj2 - The second vector.

- CreatePossibleVectors

  public **S**ystem.Collections.Generic.List{BMPAnalysis.Vector} **C**reatePossibleVectors( )

  Creates a list of all possible vectors that can be created with the given amount of colors.

  - **P**arameters
    - colors - The possible colors that can, but may not appear in each vector.
    - dimX - The width of the pattern. The size of the vector will be the product of the given width and height of the pattern.
    - dimY - The height of the pattern. The size of the vector will be the product of the given width and height of the pattern.

- *E*quals

  public **b**ool **E**quals( )

  Determines whether the specified Vector is equal to the current Vector.

  - **P**arameters

    * obj - The Vector to compare with the current Vector.

- *S*wap

  public **b**ool **S**wap( )

  Swaps the colors of two positions in the vector array.

  – **P**arameters

    * pos1 - The first position in the vector array to be swapped with the second position.
    * pos2 -

- *T*oString

  public **s**tring **T**oString( )

  Returns a (a vector string) that represents the current .

Appendix D

Namespace BMPBuilder

*N*amespace Contents Page

Classes

D.1 Classes

D.1.1 Class Bitmap

The Bitmap class defines basic properties and functions for a square-shaped checkboard-like image. Creating, Painting, Loading and Saving of images is possible. This class should not be instantiated directly - its inherited classes Picture and Pattern should be used instead.

Declaration

```
public class Bitmap
    : Object
```

Properties

- Colors

  public **S**ystem.Collections.Generic.List{System.Drawing.Color} **C**olors { get; set; }

  A list of all colors the Image contains. (E.g., if the Image contains squares in black and white, this List will contain two color objects with the values black and white.)

- *S*quaresX

  public **i**nt **S**quaresX { get; set; }

  Numbers of squares in horizontal directions

- *S*quaresY

  public **i**nt **S**quaresY { get; set; }

  Numbers of sqares in vertical direction

- *S*tartLocation

  public **S**ystem.Drawing.Point **S**tartLocation { get; set; }

The X and Y coordinate the image is displayed on the screen. Important to calcuate a square Position from a given point on the screen.

- *S*tructure

  public **S**ystem.Int32[,] **S**tructure { get; set; }

  The structure of the image, defined in a two-dimensional array of integer values.

Constructors

- .ctor

  public **B**itmap( )

  Creates a new image object.

  – **P**arameters
    * squaresX - The number of squares in horizontal direction.
    * squaresY - The number of squares in vertical direction.
    * colors - A list of colors which should be used for the image. (E.g, black and white)

Methods

- ColorDistribution

  public **S**ystem.Double[] **C**olorDistribution( )

  Returns the percentage distribution of colors of an (already created) image. E.g, if a 2x2 image consists of 1 black square and 3 white squares, the distribution would be 0.25 and 0.75.

  – **P**arameters
    * decimals - Truncates the result to the specified decimal places

- ColorDistribution

  public **S**ystem.Double[] **C**olorDistribution( )

Returns the percentage distribution of colors of the defined clipping area of an image. E.g, if the defined clipping area spans over a 2x2 pattern and consists of 1 black square and 3 white squares, the distribution would be 0.25 and 0.75.

  - **P**arameters
    * `d`ecimals - Truncates the result to the specified decimal places
    * `d`istRect - A clipped area defined as a rectangle. Should not exceed the size of the image.

- *C*reate

  `p`ublic **`v`**`oid` **C**reate( )

  Fills the image at random with the colors defined in the Color Property

- *L*oad

  `p`ublic **`B`**`MPBuilder.Bitmap` **L**oad( )

  Loads previously saved image data.

  - **P**arameters
    * `f`ilename - The path and filename of a previously saved object.

- *P*aint

  `p`ublic **`S`**`ystem.Drawing.Bitmap` **P**aint( )

  Returns a ”real” bitmap of the type System.Drawing.Bitmap of the created image.

  - **P**arameters
    * `s`quareSize - The Width and Height of each Square in Pixel
    * `b`orderColor - The color of the surrounding border

- *S*ave

  `p`ublic **`v`**`oid` **S**ave( )

  Saves the image data to a file on the hardisk.

  - **P**arameters
    * `f`ilename - The path and filename where to store the object.

D.1.2 Class Constants

The Constants class defines fixed values for the experiments, such as the screen size or the maximum number of squares for a pattern.

Declaration

```
public class Constants
    : Object
```

Properties

- *B*orderSize

  public **i**nt **B**orderSize { get; set; }

  Gets or sets the size of the rectangle frame that marks the target squares at the end of each trial.

- *C*onfigFileName

  public **s**tring **C**onfigFileName { get; }

  Gets the name of the config file.

- *D*ataFolderName

  public **s**tring **D**ataFolderName { get; }

  Gets the name of the data folder.

- *D*ataPostfix

  public **s**tring **D**ataPostfix { get; }

  Gets the data file ending (e.g.,set to .bin)

- *D*isplayX

  public **i**nt **D**isplayX { get; }

  Gets the horizontal resolution of the experiment screen.

- *D*isplayY

  public **i**nt **D**isplayY { get; }
  Gets the vertical resolution of the experiment screen.

- *E*xperimentFileName

  public **s**tring **E**xperimentFileName { get; }
  Gets the name of the experiment file.

- *I*mageFolderName

  public **s**tring **I**mageFolderName { get; }
  Gets the name of the image folder.

- *M*axPatternsX

  public **i**nt **M**axPatternsX { get; }
  Gets the maximum number of pattern squares in horizontal direction.

- *M*axPatternsY

  public **i**nt **M**axPatternsY { get; }
  Gets the maximum number of pattern squares in vertical direction.

- *M*axSizeX

  public **i**nt **M**axSizeX { get; }
  Gets the maximum width for a square.

- *M*axSizeY

  public **i**nt **M**axSizeY { get; }
  Gets the maximum height for a square.

- *P*atternDataPrefix

  public **s**tring **P**atternDataPrefix { get; }

Gets the pattern data prefix.

- *P*atternHeight

  public **i**nt **P**atternHeight { get; }

- *P*atternNumberBeforeTimeChange

  public **i**nt **P**atternNumberBeforeTimeChange { get; }

  Gets the number of trials done before display time changes for first time.

- *P*atternNumberBeforeTimeChange2

  public **i**nt **P**atternNumberBeforeTimeChange2 { get; }

  Gets the number of trials done before display time changes for second time.

- *P*atternOffset

  public **i**nt **P**atternOffset { get; }

  Gets the vertical (X-Axis) start position of the pattern display.

- *P*atternWidth

  public **i**nt **P**atternWidth { get; }

  Gets the maximum width for a pattern object.

- *P*ictureDataPrefix

  public **s**tring **P**ictureDataPrefix { get; }

  Gets the picture data prefix.

- *P*ictureHeight

  public **i**nt **P**ictureHeight { get; }

  Gets the maximum height for a picture object.

- *P*ictureWidth

public **i**nt **P**ictureWidth { get; }

Gets the maximum width for a picture object.

Constructors

- .ctor

  public **C**onstants( )

Methods

- CalcPatternLocation

  public **S**ystem.Drawing.Point **C**alcPatternLocation( )

  Calculates the pattern location on the screen.

  – **P**arameters

  * sameScreen - If set to true, pattern and picture are displayed on the same screen.
  * patternSquaresX -
  * sizeX - The size of each square (in pixel) in horizontal direction.
  * patternSquaresY -
  * sizeY - The size of each square (in pixel) in vertical direction.

- CalcPictureLocation

  public **S**ystem.Drawing.Point **C**alcPictureLocation( )

  Calculates the picture location on the screen.

  – **P**arameters

  * sameScreen - If set to true, pattern and picture are displayed on the same screen.
  * squaresX - The number of squares in horizontal direction.
  * sizeX - The size of each square (in pixel) in horizontal direction.
  * squaresY - The number of squares in vertical direction.
  * sizeY - The size of each square (in pixel) in vertical direction.

- ColorIntToString

  public **s**tring **C**olorIntToString( )

  Makes a string out of a color object.

  - **P**arameters
    * colValue - The color value as an object.
    * abbr - If set to true, an abbreviated form will be used.

- ColorIntToString

  public **s**tring **C**olorIntToString( )

  Makes a string out of an integer color value.

  - **P**arameters
    * colValue - The color value as an integer.
    * abbr - If set to true, an abbreviated form will be used.

D.1.3 Class ExperimentData

Class to store and handle all necessary information for an experiment. It contains a list of all Trials as well as functions to load and save the experiment and generate data files for the experiment engine.

Declaration

```
public class ExperimentData
    : Object
```

Properties

- *T*rials

  public **S**ystem.Collections.Generic.List{BMPBuilder.TrialData} **T**rials { get; set; }

  Gets or sets the list of trials.

Constructors

- .ctor

  public **E**xperimentData( )

Methods

- *G*enerateExperimentFile

  public **b**ool **G**enerateExperimentFile( )

  Generates the experiment file for the experiment engine..

  – **P**arameters

  * folder - The folder where the files are to be created.
  * worker - A background worker object that runs this function in a background thread and can retrieve progress information.

- *L*oadExperiment

  `p`ublic **`B`**`MPBuilder.ExperimentData` **L**oadExperiment( )

  Loads the experiment.

  - **P**arameters
    - `f`ilename - The filename (including path) of the experiment to be loaded.

- *S*aveExperiment

  `p`ublic **`v`**`oid` **S**aveExperiment( )

  Saves the experiment.

  - **P**arameters
    - `f`ilename - The filename (including path) of the experiment.

D.1.4 Class ExperimentForm

The GUI for the Experiment Builder.

Declaration

```
public class ExperimentForm
    : Form
```

Properties

- *E*xperiment

  public **B**MPBuilder.ExperimentData **E**xperiment { get; set; }

  Gets or sets the experiment data.

Constructors

- .ctor

  public **E**xperimentForm( )

  Initializes a new instance of the class.

Methods

- *U*pdateTrialList

  public **v**oid **U**pdateTrialList( )

  Updates the trial list.

D.1.5 Class Info

Displays the Info window with description about the author and program.

Declaration

```
public class Info
    : Form
```

Constructors

- .ctor

  public **I**nfo( )

D.1.6 Class Pattern

The Pattern class inherits from the Bitmap class, and represents a pattern image.

Declaration

```
public class Pattern
    : Bitmap
```

Constructors

- .ctor

  public **P**attern( )

  Initializes a new instance of the class.

  – **P**arameters

    * squaresX - The number of squares in horizontal direction.
    * squaresY - The number of squares in vertical direction.
    * colors - A list of colors which should be used for the image. (E.g, black and white)

D.1.7 Class Picture

The Picture class inherits from the Bitmap class, and represents a picture in which a target pattern can be found.

Declaration

```
public class Picture
    : Bitmap
```

Properties

- *P*atternPositionX

  public **i**nt **P**atternPositionX { get; set; }

  Gets or sets the pattern position X-coordinate (horizontal).

- *P*atternPositionY

  public **i**nt **P**atternPositionY { get; set; }

  Gets or sets the pattern position Y-coordinate (vertical).

Constructors

- .ctor

  public **P**icture( )

  Initializes a new instance of the class.

  - **P**arameters
    - squaresX - The number of squares in horizontal direction.
    - squaresY - The number of squares in vertical direction.
    - colors - A list of colors which should be used for the image. (E.g, black and white)

Methods

- CalculatePosX

  `public int CalculatePosX( )`

  Calculates the horizontal position in the structure of a given pixel location on the screen. Depends on the start location and the size of the squares.

  - **P**arameters
    - `averageX` - The pixel location (X-Axis/horizontal) on the screen.
    - `sizeX` - The horizontal size of each square.

- CalculatePosY

  `public int CalculatePosY( )`

  Calculates the vertical position in the structure of a given pixel location on the screen. Depends on the start location and the size of the squares.

  - **P**arameters
    - `averageY` -
    - `sizeY` -

- *I*sPatternFixation

  `public bool IsPatternFixation( )`

  Determines whether the given position in the structure is a pattern fixation. Includes the surrounding squares - the radius specifies how many squares are included.

  - **P**arameters
    - `pos` - The position in the structure.
    - `radius` - The radius, specifies how many surrounding squares are included. A radius of 0 includes only the minimun squares, which depends on the pattern size. The bigger the radius, the more squares (also depending on the pattern size) are included.
    - `includePatternFixation` - If set to true the function always returns false.

- *I*sPatternFixation

  public **b**ool **I**sPatternFixation( )

  Determines whether the given position in the structure is a pattern fixation. Includes the surrounding squares - the radius specifies how many squares are included.

  - **P**arameters
    - pos - The position in the structure.
    - radius - The radius, specifies how many surrounding squares are included. A radius of 0 includes only the minimun squares, which depends on the pattern size. The bigger the radius, the more squares (also depending on the pattern size) are included.

- *R*emoveAndAddPattern

  public **v**oid **R**emoveAndAddPattern( )

  Removes and adds a pattern to the picture.

  - **P**arameters
    - pattern - The pattern to be added.

D.1.8 Class PreviewData

Class to store and handle all necessary information of a trial which is needed to generate a preview of a trial.

Declaration

```
public class PreviewData
    : Object
```

Properties

- *B*ackgroundColor

  public **S**ystem.Drawing.Color **B**ackgroundColor { get; set; }

  Gets or sets the color of the background.

- *B*orderColor

  public **S**ystem.Drawing.Color **B**orderColor { get; set; }

  Gets or sets the color of the border.

- *C*olor1

  public **S**ystem.Drawing.Color **C**olor1 { get; set; }

  Gets or sets the color 1.

- Color2

  public **S**ystem.Drawing.Color **C**olor2 { get; set; }

  Gets or sets the color 2.

- Color3

  public **S**ystem.Drawing.Color **C**olor3 { get; set; }

  Gets or sets the color 3.

- Color3Checked

  public **b**ool **C**olor3Checked { get; set; }
  Gets or sets a value indicating whether color 3 is activated.

- Color4

  public **S**ystem.Drawing.Color **C**olor4 { get; set; }
  Gets or sets the color 4.

- Color4Checked

  public **b**ool **C**olor4Checked { get; set; }
  Gets or sets a value indicating whether color 4 is activated.

- Color5

  public **S**ystem.Drawing.Color **C**olor5 { get; set; }
  Gets or sets the color 5.

- Color5Checked

  public **b**ool **C**olor5Checked { get; set; }
  Gets or sets a value indicating whether color 5 is activated.

- Color6

  public **S**ystem.Drawing.Color **C**olor6 { get; set; }
  Gets or sets the color 6.

- Color6Checked

  public **b**ool **C**olor6Checked { get; set; }
  Gets or sets a value indicating whether color 6 is activated.

- ConclusionColor

  public **S**ystem.Drawing.Color **C**onclusionColor { get; set; }

Gets or sets the color of the rectangular frame the marks the target pattern in the picture at the end of a trial.

- *D*isplayX

  public **i**nt **D**isplayX { get; set; }

  Gets or sets the width of the display - should be the screen width (in pixel) of the experiment pc.

- *D*isplayY

  public **i**nt **D**isplayY { get; set; }

  Gets or sets the height of the display - should be the screen width (in pixel) of the experiment pc.

- *P*atternDisplayTime

  public **i**nt **P**atternDisplayTime { get; set; }

  Gets or sets the pattern display time for the first trials.

- *P*atternDisplayTime2

  public **i**nt **P**atternDisplayTime2 { get; set; }

  Gets or sets the pattern display time after a specified number of trials.

- *P*atternDisplayTime3

  public **i**nt **P**atternDisplayTime3 { get; set; }

  Gets or sets the pattern display time after a specified number of trials.

- *P*atternSquaresX

  public **i**nt **P**atternSquaresX { get; set; }

  Gets or sets the number of squares in horizontal direction for the pattern.

- *P*atternSquaresY

  public **i**nt **P**atternSquaresY { get; set; }

  Gets or sets the number of squares in vertical direction for the pattern.

- *S*ameScreen

  public **b**ool **S**ameScreen { get; set; }

  Gets or sets a value indicating whether the pattern and the picture are displayed on the same screen.

- *S*izeX

  public **i**nt **S**izeX { get; set; }

  Gets or sets the width of each square (in pixel).

- *S*izeY

  public **i**nt **S**izeY { get; set; }

  Gets or sets the height of each square (in pixel).

- *S*quaresX

  public **i**nt **S**quaresX { get; set; }

  Gets or sets the number of squares in horizontal direction in a trial.

- *S*quaresY

  public **i**nt **S**quaresY { get; set; }

  Gets or sets the number of squares in vertical direction in a trial.

Constructors

- .ctor

  public **P**reviewData( )

D.1.9 Class PreviewForm

This class displays a rough preview of a pattern and picture on the screen.

Declaration

```
public class PreviewForm
    : Form
```

Constructors

- .ctor

  public **P**reviewForm( )

Methods

- CreatePreviewForm

  public **v**oid **C**reatePreviewForm( )

  Creates the preview form.

  – **P**arameters

    * preview - The preview data.

D.1.10 Class TrialData

Class to store and handle all necessary information of a trial. All trial data which is need to generate a preview is stored the linked PreviewData object.

Declaration

```
public class TrialData
    : Object
```

Properties

- *D*uration

  public **i**nt **D**uration { get; set; }
  Gets or sets the duration of the trial.

- *N*ame

  public **s**tring **N**ame { get; set; }
  Gets or sets the name of the trial.

- *P*review

  public **B**MPBuilder.PreviewData **P**review { get; set; }
  Gets or sets all trial information needed to generate a preview.

- *T*rialNumbers

  public **i**nt **T**rialNumbers { get; set; }
  Gets or sets the number of trials.

Constructors

- .ctor

  public **T**rialData( )

D.1.11 Class TrialForm

This class is the GUI for entering trial information.

Declaration

```
public class TrialForm
    : Form
```

Constructors

- .ctor

  public **T**rialForm( )

  Initializes a new instance of the class.

- .ctor

  public **T**rialForm( )

  Initializes a new instance of the class.

  – **P**arameters

    * experimentForm - The experiment form.

- .ctor

  public **T**rialForm( )

  Initializes a new instance of the class.

  – **P**arameters

    * experimentForm - The experiment form.
    * editNumber - The edit number of the trial to be changed.

List of Symbols and Abbreviations

Abbreviation	Description	Definition
EyeTracker	Eye tracking device (e.g. EyeLink II from SR Research)	page 8
FPOI	Fixation pattern of interest	page 31
R	Radius	page 53
ROI	Region of interest	page 30

List of Figures

List of Tables