Chapter 15
The Gaze Analytics Pipeline

Often the goal of recording eye movements is to gain insight into human behavior, perception, or analysis of some performance via eye movement process measures. More often than not the data eventually need to be evaluated with respect to some hypothesis, e.g., as set forth during the design of an experiment. Evaluation of the hypothesis, in turn, usually requires some statistical manipulation which yields rejection (or failure of rejection) of a null hypothesis. While eye trackers excel at recording eye movement data, they often lack the software required to accomplish its statistical testing. Eye tracking vendors cannot be blamed for this as they can hardly be expected to anticipate all possible experimental manipulations for which their devices are used. In order to accomplish the ultimate task required by analysis, an external software package built for statistical analysis is relied upon, e.g., `R`, the software for statistical computing. To be able to use this software, the data recorded *and exported* by the eye tracker needs to be translated to a data format for input to the statistical software. This is the purpose of arranging an *analytics pipeline* as described in this chapter. The gaze analytics pipeline is largely concerned with eye movement analysis as its first step. Once basic eye movement *events* are extracted, they can be sequenced into data suitable for statistical analysis.

The *Gaze Analytics Pipeline* consists of the following goals:

1. denoise and filter raw gaze data $g_i = (x_i, y_i, t_i)$ to classify raw gaze into fixations $f_i = (x_i, y_i, t_i, d_i)$, where (x_i, y_i) coordinates indicate the position of the gaze point or centroid of the fixation, with t_i indicating the timestamp of the gaze point or fixation and d_i the fixation's duration,
2. collate fixation-related information for its subsequent statistical comparison,
3. interpret and visualize statistical tests conducted on processed data.

Visualization of the data at each stage of the pipeline is particularly helpful in fine-tuning parameters, such as threshold levels for velocity-based filtering. Filtering of the raw gaze data can be further broken down into the following operations.

A.T. Duchowski, *Eye Tracking Methodology*, DOI 10.1007/978-3-319-57883-5_15

1. Visual angle conversion, based on recorded:
   - display screen width, height (e.g., 1600×900)
   - screen dimensions (e.g., 17 in along the diagonal)
   - viewing distance D (e.g., 65 cm, or 26 in.).
2. Butterworth smoothing (optional), which requires the following parameters:
   - filter order (e.g., 2nd, 3rd, etc.)
   - sampling rate (e.g., 60, 150, 300 Hz, etc.)
   - cutoff frequency (e.g., 6.15 Hz)
3. Savitzky-Golay differentiation, based on the following parameters:
   - filter width (e.g., specified as half-width, 5)
   - degree (e.g., 3rd order for a cubic polynomial)
   - order (e.g., 1 for differentiation, 0 for smoothing)
4. Thresholding to determine saccades:
   - velocity threshold (e.g., 30 deg/s)

Recall that the viewing angle subtended by the screen is $2\tan^{-1}(\texttt{screen}/2D)$; for pixel distances, these first need to be converted to inches via a dots-per-inch calculation, dependent on the physical dimensions of the screen. The task of filtering is greatly facilitated by available signal processing libraries, e.g., `Python` wherein one can fairly quickly produce custom filtering code or use something like `Python`'s `signal` routines found in the `scipy` module. The above filtering and analytical goals are grouped into five steps usually expressed as `Makefile` targets, detailed below.

15.1 Gaze Analytics in Five Easy Steps

The goal of eye movement analysis is to identify, locate, and label *events* from raw data samples (Holmqvist et al. 2011). Of particular interest of course are fixations and saccades (and smooth pursuit movements although as Holmqvist et al. note, a robust and generic algorithm for their detection is currently an open research problem). Given readily available implementations of digital filters, e.g., Finite Impulse Response (FIR) or Infinite Impulse Response (IIR) filters in software libraries such as those of `Python`, or see also Hollos and Hollos (2014), usage of analysis techniques can be incorporated into a *gaze analytics pipeline* whose goal is convenience and automation. In this chapter an implementation of such a pipeline (in `Python`) is described, where the analytics proceeds along the following steps, as shown in Fig. 15.1:

1. `dirs`: create directories for processed data
2. `raw`: extract raw gaze data $p_i = (x_i, y_i, t_i)$

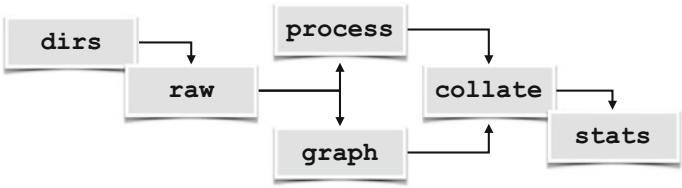


Fig. 15.1 The gaze analytics pipeline

3. `graph` or `process`: detect fixations $f_i = (x_i, y_i, t_i, d_i)$ and/or visualize
4. `collate`: assemble fixations into data files for statistical analysis
5. `stats`: perform statistical analysis

Ideally, once everything (including raw data, stimulus imagery, and above pipeline code) is in place, the entire process can be executed by one (command-line) command. On Unix-like systems (e.g., Linux or Mac), the `Makefile` utility is useful since all steps can eventually be run by typing `make`. On Windows systems, a similar effect can be achieved through the use of batch (`.bat`) files, assuming all required software is installed *a priori*. The implementation of the above pipeline described in this Chapter relies on `Python` (v2.7) and `R`, the language for statistical computing. Both `Python` and `R` are free, open-source, and available for installation on most currently popular operating systems (i.e., Windows, Linux, and Mac).

15.1.1 Step 0: Data Collection

Although eye tracking software might not possess all the tools required to complete statistical analysis, most, if not all, eye trackers provide some mechanism to export data. Incidentally, this may be a key feature of eye tracking software when evaluating purchase of an eye tracker. Some export various eye movement events, e.g., fixations, Areas Of Interest, time to first fixations, etc. One compulsory attribute should also be raw gaze data. Raw gaze data allows application of custom eye movement analysis filters. Why should this be beneficial is discussed below.

Most eye trackers are adept at maintaining recorded data in some kind database or flat file organization that lends itself to exportation. Usually the eye tracker will export data in a format that could be organized as one file per subject. Each file may then contain all of the gaze data collected over all stimuli that the participant looked at in the particular order presented to them. To begin with, these exported data files need to be parsed and "exploded" into files that essentially contain one scanpath per file. A good file naming scheme is important here. The file name itself can serve as an indicator of participant ID and condition, e.g.,

```
10_test_mid-1.raw  10_test_mid-2.raw  10_test_mid-3.raw
11_test_low-1.raw  11_test_low-2.raw  11_test_low-3.raw
...
```

would indicate (at least) three trials (`-1`, `-2` and `-3`) in each of two conditions (`mid` and `low`). for two participants (`10` and `11`). Using the file name to encode the experimental trials facilitates sorting as well as parsing for collation of data in subsequent steps. The above `.raw` files are the results of Step 2 below.

Before starting on the description of the analytics pipeline, assume that all data from the experiment is collected in some directory. This may mean a large number of files, one per participant in the eye tracking study, as exported by the eye tracking software.

15.1.2 Step 1 (`dirs`): Directory Creation

This step is exceptionally straightforward since it only relies on creating subdirectories for raw data to be written to. However, at this point it is instructive to examine Listing 15.1 which shows the top of a `Makefile` used for the pipeline.

The `dirs` target simply sets up the subdirectories for collection of extracted raw data files and is the first target of the pipeline. Above this target several variables are initialized specifying conditions under which eye movement data was collected, e.g., width and height of the monitor (from a table-mounted run in this case), sampling rate (e.g., 60 Hz), average viewing distance to the screen, and physical (diagonal) dimension of the screen (for conversion of gaze data distances to degrees visual angle).

A benefit of setting up subdirectories for raw (and processed) data files to be written to is so that these directories can be deleted *en masse* when not needed (e.g., to save disk space). The `Makefile`'s `clean` target removes these directories.

15.1.3 Step 2 (`raw`): Extract Raw Gaze Data

Prior to eye movement analysis, the data files need to be parsed so that each trial is extracted into its own file, if it is not already. At this point the data may also undergo denoising, conversion, or normalization. At this point data pertaining to blinks may also be removed.

Given raw gaze data as exported by an eye tracker, such as what is excerpted in Listing 15.2, this step of the pipeline merely extracts the raw gaze data and timestamp $p_i = (x_i, y_i, t_i)$ from the exported file into a file just containing those three elements into a file stored in the directories created in the first step. The `Makefile` in Listing 15.1 uses the `csv2raw.py` `Python` script for performing this data translation.

Although this step seems redundant, there are several reasons for doing this. First, the exported data may reside in an entirely different location from where the processing scripts may be. This establishes a separation between data and code. Generally data captured by the eye tracker should not be manipulated in any way but

```
PYTHON = python2

WIDTH  = 1600   # width in pixels
HEIGHT = 900    # height in pixels
HERTZ  = 60     # sampling rate
DIST   = 22.44  # 57 cm
SCREEN = 17     # diagonal screen dimension in inches

# use Butterworth filter?
SMOOTH = False

XTILES = 2      # for uniform AOI grid
YTILES = 3      # for uniform AOI grid

# where data and images live
INDIR  = ../../exp/pilot/etdata/
IMGDIR = ../../exp/pilot/shots/

PLTDIR = ./plots/
OUTDIR = ./data/
RAWDIR = ./data/raw/

all: dirs raw process collate stats

dirs:
        mkdir -p data
        mkdir -p data/raw
        mkdir -p plots

raw:
        $(PYTHON) ./csv2raw.py --indir=$(INDIR) \
                               --outdir=$(RAWDIR)
...
```

Listing 15.1 Example top section of `Makefile`

```
cnt,time,fpogx,fpogy,fpogs,fpogd,...
45002.0,3633.72607,0.37806,0.38487,3633.46362,0.26245,...
45003.0,3633.74243,0.37968,0.38454,3633.46362,0.27881,...
45004.0,3633.75879,0.38077,0.38305,3633.46362,0.29517,...
45005.0,3633.77539,0.37864,0.35665,3633.46362,0.29517,...
45006.0,3633.79272,0.38445,0.38529,3633.46362,0.29517,...
...
```

Listing 15.2 Excerpt from tracker export (file formats differ among vendors)

should be preserved unaltered. Therefore, it makes sense to extract just enough data that is needed without touching empirical data in any way.

Second, if one is sharing the data and code among collaborators, processed data is generally transient or temporary in nature and thus no compunction should be left when deleting temporary data. Indeed, in the interests of disk usage and synchronization between collaborators, it is best to share as little as possible thereby minimizing storage and transmission.

Third, in the interest of code re-use, if processing scripts operate on a common and simple data set, i.e., a file containing only a list of three values (x_i, y_i, t_i), then there is no need to rewrite the code. If eye tracker data is later obtained in a different format, as from another tracker, for example, then only this intermediate data translation step needs to be rewritten to extract the raw gaze data so that subsequent processing steps remain unaltered.

```
0.386540  0.340390  3633.726070
0.407310  0.379050  3633.742430
0.400360  0.356180  3633.758790
0.402180  0.364200  3633.775390
0.390260  0.413920  3633.792720
0.415250  0.342320  3633.808110
0.414480  0.338520  3633.825440
0.436120  0.406820  3633.841060
0.419500  0.322960  3633.857420
...
```

Listing 15.3 Raw data extracted from tracker export

An example of raw data extracted from eye tracker data is shown in Listing 5.3. Notice that the raw gaze data is normalized with the timestamp expressed in seconds. The (average or expected) difference between successive timestamps in this example is $3633.742430 - 3633.726070 = 0.016$ s, indicating a 60 Hz sampling rate (the `Makefile` in Listing 15.1).

Writing this type of *translator* script presents a good opportunity to ensure that the extracted raw data conforms to expectations of being normalized with the timestamp expressed in seconds. If the data is not originally provided in this format, it could be normalized in this step, with the timestamp adjusted so that the first entry is set to 0.000000, if need be.

15.1.4 *Step 3 (*`graph` *or* `process`*): Graph or Process Raw Data*

Given raw data, a strong temptation is to process the data to then be able to jump ahead to collation and statistical analysis. Resist this urge. Instead, use visualization

as a form of sanity check. Although time-consuming and prone to generation of a large number of possibly large files, in this step it is worthwhile producing graphs of the data, e.g., scanpaths or heatmaps. Heatmaps are generally slower to render and perhaps can be avoided at the outset. Scanpaths, however, can reveal several problems in data translation above. For example, it is important to double-check image (stimulus) and screen dimensions, screen resolution, as well as location of the origin in the exported gaze data. Usually the origin (0, 0) can be expected at the top-left of the image. Drawing the data, however, may need flipping of the y-coordinate as typical graphics systems assume the origin at bottom-left.

```
...

raw:
        $(PYTHON) ./csv2raw.py --indir=$(INDIR) \
                               --outdir=$(RAWDIR)

process:
        $(PYTHON) ./filter.py --smooth=$(SMOOTH) \
                  --indir=$(RAWDIR) --imgdir=$(IMGDIR) \
                  --dist=$(DIST) --screen=$(SCREEN) \
                  --width=$(WIDTH) --height=$(HEIGHT) \
                  --hertz=$(HERTZ) \
                  --xtiles=$(XTILES) --ytiles=$(YTILES)

graph:
        $(PYTHON) ./graph.py --smooth=$(SMOOTH) \
                  --indir=$(RAWDIR) --imgdir=$(IMGDIR) \
                  --dist=$(DIST) --screen=$(SCREEN) \
                  --width=$(WIDTH) --height=$(HEIGHT) \
                  --hertz=$(HERTZ) \
                  --xtiles=$(XTILES) --ytiles=$(YTILES) \
                  --outdir=$(OUTDIR) --pltdir=$(PLTDIR)

collate:
        $(PYTHON) ./collate-amfo.py
        $(PYTHON) ./collate-fxtn.py
        $(PYTHON) ./collate-aois.py

stats:
        $(R) --vanilla < amfo.R > amfo.out
        $(R) --vanilla < fxtn.R > fxtn.out
        $(R) --vanilla \
             --args $(XTILES) $(YTILES) < tm.R > tm.out

clean:
        rm -rf *.pyc *.pdf *.out
        rm -rf plots/ data/
```

Listing 15.4 Example bottom section of `Makefile`

Note that the graph step is composed of the event (fixation) detection along with graphics step. That is, the script used to process the data is basically a stripped-down version of the script used in the graph step. In Listing 15.4 depicting the bottom section of the `Makefile`, scripts `graph.py` and `filter.py` perform the graphics and process steps, respectively. Generally, if the visualizations *appear* reasonable, they will instill confidence in the process step which is used to produce metrics of interest, e.g., numbers of fixations, or fixations in Areas Or Interest, AOIs.

The graph step can also be used to compare fine-tuning of digital filters either against a calibration step (preferred) or against visualizations produced by the eye tracking vendor. Figure 15.2 shows a comparison between scanpaths generated by a vendor's software and by custom pipeline software. Regardless of the filter used by the vendor (which some might not wish to disclose, limiting replication by others), it is clear that the filter used in the pipeline matches fixation locations with what the vendor obtained.

Visualization can sometimes improve upon whatever is provided by the vendor. Figure 15.2 shows fixations with disc radii made relative to fixation durations: larger discs indicate relatively longer fixation durations. The custom rendering also depicts arrowheads which indicate sequential ordering of fixations which is lacking in the vendor's visualization.

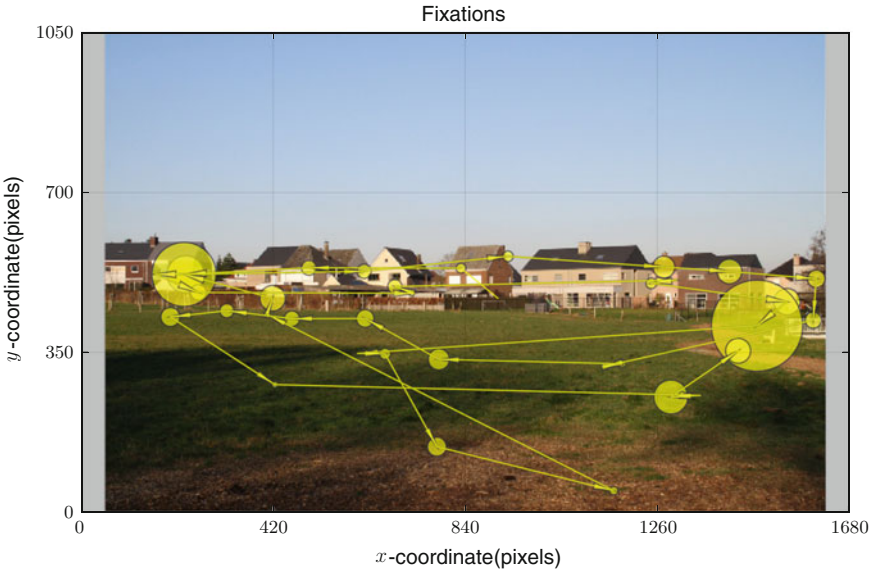
Once satisfied with visualizations, the process step is then used to extract fixations (or other relevant events, e.g., microsaccades, pupillometric data, etc.) from each data file for subsequent collation and statistical analyses. Being stripped of graphics rendering routines, this step typically runs much faster than the graph step.

```
3633.7587 657.2833 338.9073 0.4108 380.3139 0.1149
3634.2846 1010.8925 198.9102 0.2465 250.0825 0.0493
3634.5805 798.9905 331.7214 1.6921 189.7252 0.0986
3636.3713 984.1169 373.2411 0.6242 213.2928 0.0983
3637.0939 1195.5960 345.4837 0.2465 281.0482 0.1147
3637.4553 915.8986 373.0065 0.3288 300.7593 0.0822
3637.8664 1214.8206 406.2008 1.0683 11.5830 0.0156
3638.9504 1223.7600 398.8350 0.0163 469.6898 0.0988
...
```

Listing 15.5 Fixation data extracted from raw data into a `-fxtn.dat` file, made up of timestamp, x, y, duration, saccade amplitude, and saccade duration

An example of the type of data produced by the process step is shown in Listing 15.6. In this instance, fixation data contains the fixation start timestamp, its (x, y) centroid coordinates, duration, amplitude (distance) of the subsequent saccade, and the subsequent saccade duration. What is important to note here is that this data is stored in its own file, i.e., one per specific condition for each participant in the experiment. Another such set of data files may also exist for the AOIs processed, as well as for the ambient/focal fixation information. It is the job of the next step to collate these individual files into one one for subsequent statistical analysis.

(a) Vendor-provided scanpath visualization.



(b) Gazedata processed via analytics pipeline.

Fig. 15.2 Eye movement signal processing via analytics pipeline. Courtesy of Lien Dupont

```
subj,block,task,stim,timestamp,x,y,dur,sacc_amp,sacc_dur
10,test,high,1,3633.7587,657.2833,338.9073,0.4108,...
10,test,high,1,3634.2846,1010.8925,198.9102,0.2465,...
10,test,high,1,3634.5805,798.9905,331.7214,1.6921,...
10,test,high,1,3636.3713,984.1169,373.2411,0.6242,...
10,test,high,1,3637.0939,1195.5960,345.4837,0.2465,...
...
```

Listing 15.6 Collated fixation data assembled from `-fxtn.dat` files

15.1.5 Step 4 (`collate`): Collate Data Prior to Statistical Analysis

Listing 15.4 contains three scripts used to collate processed data: `collate-amfo.py`, `collate-fxtn.py`, and `collate-aois.py`, which assemble together files containing information pertaining to ambient/focal fixations (see Chap. 14), fixations, and Areas Of Interest (AOIs).

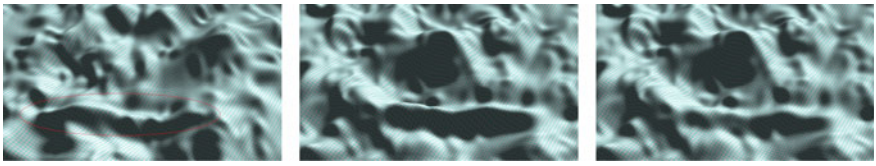
The main task of each collate script is to assemble all like data from every participant under each experimental condition into one file. For example, all fixation data (stored in `-fxtn.dat` files) are assembled, or collated, into a `fxtn.csv` file. Similarly, all `-aois.dat` data are collated into a `aois.csv` file and all `-amfo.dat` are collated into a `amfo.csv` file.

The collate scripts are fairly straightforward: they merely copy all the information from each data file into the larger comma-separated-value file. In doing so, each line of data is prepended by its identifying information, consisting of, for example, participant number, block, task, and stimulus. An example of collated fixation data is shown in Listing 15.6. This file's distinguishing feature is its header line, which serves to label the columns of data for subsequent statistical analyses.

15.1.6 Step 5 (`stats`): Perform Statistical Analyses

Statistical analysis, perhaps relying on `R`, the language for statistical computation, or other statistical software, is performed on the collated data. Collated data contains related information from all participants and from all experimental conditions. If using `R`, generally the collated files will contain *one observation per line*, and not data pertaining to *one participant per line*, as is perhaps more intuitive.

Statistical analyses generally consist of analysis of variance (ANOVA) on the dependent variables collected during the eye tracking experiment, e.g., fixation durations, fixation counts, etc., depending of course on the experimental design and its hypotheses. Along with statistical tests of inference, plots are generated to visualize the results.



(a) Target shown (training). (b) Target high (easy). (c) Target low (difficult).

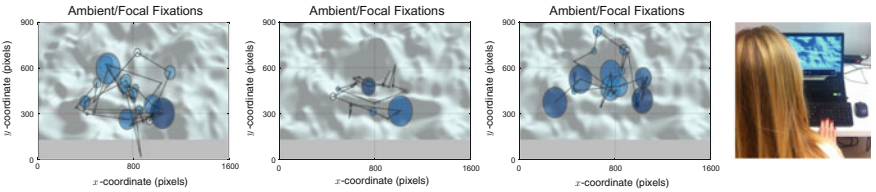
Fig. 15.3 Example stimuli from an experiment designed to capture visual search of terrain where the target surface feature is shown at varying elevations. **a** shows the target circled by an *ellipse* for training purposes; with target elevation high (**b**) (easy task); and target elevation low (**c**) (difficult task)

Organization of the statistical code can be made to match previous steps, e.g., maintaining similar file naming conventions. For example, if fixation data is collated by a `collate-fxtn.py` script and assembled into a `fxtn.csv` file, then the statistical code can be kept in a `fxtn.R` file, distinguishing it from other statistical analyses pertaining to ambient/focal fixations (`aois.R`) and transition entropy (`tm.R`). Listing 15.4 contains the three `R` invocation. Output of the analysis is redirected to corresponding `.out` files. A worked example of the analytics pipeline follows (Fig. 15.3).

15.2 Gaze Analytics: A Worked Example

The gaze analytics pipeline was employed in a straightforward visual search task. In this task, with stimuli shown in Fig. 15.4, participants were asked to locate an elevated terrain feature embedded in a (Gaussian) surface. The experimental design was a repeated measures factorial design with terrain feature serving as the fixed factor at four levels: low, mid, high elevation, of absent.

Our assumption was that these elevations would result in varying levels of (visual search) difficulty. We thus hypothesized that task difficulty would be reflected in eye movement metrics, including number of fixations, fixation durations, saccade amplitude, ambient/focal fixations, and transition entropy. Moreover, we expected that visual search would follow Just and Carpenter's (1976) three-stage model of cognitive processing: search → decision → confirmation, where the latter two steps comprise the decision-making aspect of cognition. Concordantly, we expected that eye movement measures would manifest significant responses during the decision-making aspect of the task, at the point where visual search completes and the participant must decide whether s/he has identified the feature. Just and Carpenter noted that eye fixation data make it possible to distinguish the three stages of performance, although their analysis relied on the relation between fixation duration and angular disparity. While qualitatively effective, the relation provided no easy way of combining fixation duration and disparity into a useful quantity with which to distinguish the cognitive stages. To better detect these inter-stage transitions, we apply our $\mathcal{K}$ metric



(a) Training. (b) Easy. (c) Difficult. (d) Setup.

Fig. 15.4 Fixations from a single participant's visual search of terrain elevation during training (**a**), where the target surface feature was circled by an ellipse; with target elevation high (**b**), easy task; and target elevation low (**c**), difficult task. Note that the easy task is distinguished by shorter saccades and ambient fixations, whereas the difficult task is distinguished by longer saccades and focal fixations. The *grey band* of pixels at bottom of the stimulus is typically obscured by a laptop-mounted eye tracker (**d**). Courtesy of Justyna Żurawska, SWPS University, Warsaw, Poland

(Krejtz et al. 2016), where $\mathcal{K} < 0$ refers to the situation when relatively short fixations are followed by relatively long saccades, suggesting ambient processing during visual search, and $\mathcal{K} > 0$ indicates relatively long fixations followed by short saccade amplitudes, suggesting focal processing during decision-making. Subsequent gaze transitions indicate confirmation, as noted by Just and Carpenter.

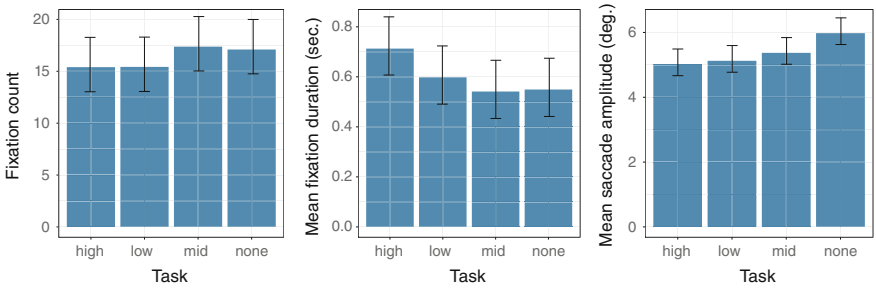
Twelve participants took part in the study, conducted at SWPS University, Warsaw, Poland. Data from four participants was dropped due to poor calibration or inattention to the task (as observed by the experimenter). Briefly, the procedure for the experiment was comprised of two blocks: training and the test block. During training, participants were shown the stimuli with the target terrain feature highlighted as a means of indicating what they were asked to visually search for.

Analysis of gaze recorded during the test task is given below in terms of traditional eye movement metrics, e.g., fixation count, duration, saccade amplitude, as well as advanced analysis of ambient/focal attention and transition entropy.

15.2.1 Scanpath Visualization

Following the gaze analytics pipeline described above, which, with the exception of Fig. 15.2, features data from the Gaussian terrain visual search experiment. Resisting the urge to dive straight into statistical analysis, visualization of processed gaze (e.g., Step 3) is shown in Fig. 15.4, along with an "over-the-shoulder" depiction of the experimental setup.

Using ambient/focal visualization (Duchowski and Krejtz 2015), Fig. 15.4 shows ambient fixations in a lighter shade of color than focal fixations. The more difficult task shows typically greater dispersion of fixations, i.e., larger saccadic jumps during visual search, hence a more ambient type of search than what is observed under easy conditions. Examining such visualizations (i.e., one per scanpath), it would appear that perhaps some of the eye movement metrics may show statistically significant differences between conditions.



(a) Fixation count. (b) Fixation duration. (c) Saccade amplitude.

Fig. 15.5 Analysis of fixation count, duration, and saccade amplitude

15.2.2 Traditional Eye Movement Metrics

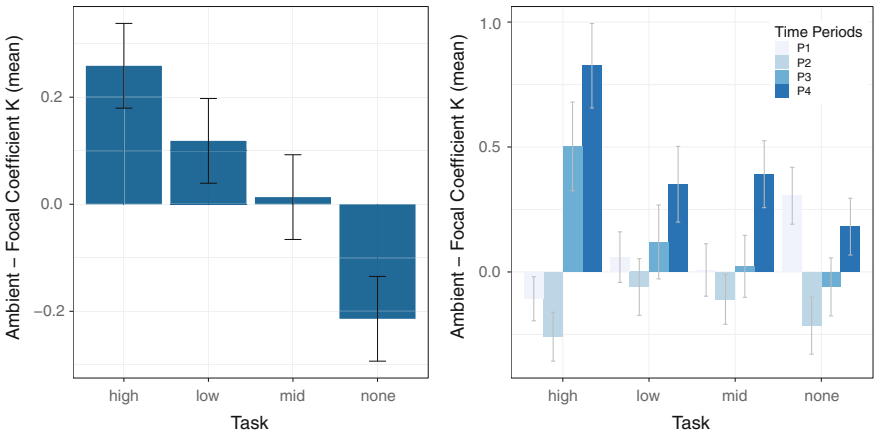
A within-subjects analysis of variance of the number of fixations shows no significant difference in their count between tasks ($F(1,3) = 0.99$, $p = 0.78$, $n.s.$, see Fig. 15.5a). In each of the tasks, approximately 15 fixations were made, on average, during visual search.

Of all fixations collected, the mean fixation duration computed was 0.54 s. A within-subjects analysis of variance of the fixation duration fails to detect a significant difference between tasks ($F(1,3) = 1.52$, $p = 0.24$, $n.s.$, see Fig. 15.5b).

A within-subjects ANOVA of saccade amplitude shows a statistically significant difference between tasks ($F(1,3) = 4.93$, $p < 0.01$, see Fig. 15.5c). Post-hoc pairwise analyses show significant differences between the control (no target) task ($\mathrm{M} = 5.98°$, $\mathrm{SE} = 0.41°$) and each of the low and ($\mathrm{M} = 5.13°$, $\mathrm{SE} = 0.41°$) high elevation ($\mathrm{M} = 5.02°$, $\mathrm{SE} = 0.41°$) tasks ($p < 0.05$ in both cases). This main effect of task suggests that when the target is absent, the visual angle between saccades is largest, which would suggest a more ambient type of search. Smaller-amplitude saccades made during the high and low elevation tasks suggests that participants made smaller saccadic jumps, ostensibly because they found the target quickly and/or were more busy fixating at the target than searching for it. This may be likely due to the task being more difficult when the target is not present. Perhaps the target shown at mid-level elevation was somehow ambiguous and also facilitated greater ambient search.

15.2.3 Advanced Eye Movement Analysis

To gain further insight into visual search behavior, ambient/focal attention can be further explored by examining $\mathcal{K}$. A within-subjects ANOVA of $\mathcal{K}$ shows a statistically significant difference between tasks ($F(1,3) = 4.84$, $p < 0.05$, see Fig. 15.6a). Post-hoc pairwise analyses shows a significant difference between the control (no target) task ($\mathrm{M} = -0.21$, $\mathrm{SE} = 0.08°$) and the high elevation ($\mathrm{M} = 0.26$,



(a) Mean $\mathcal{K}$. (b) $\mathcal{K}$ dynamics.

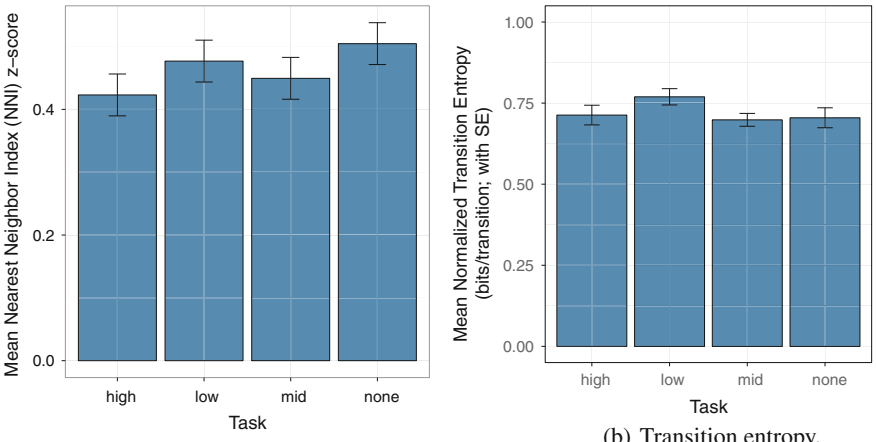
Fig. 15.6 Analysis of mean $\mathcal{K}$, $\mathcal{K}$ dynamics, and transition entropy (2×3 AOI grid)

SE $= 0.08°$) task ($p < 0.01$). $\mathcal{K} < 0$ in the target-absent tasks supports the observation that the task was significantly more ambient than the high-elevation task.

Further examination of the dynamics of $\mathcal{K}$ (see Fig. 15.6b) shows that ambient/focal viewing patterns changed over time, with time divided into 4 equal epochs. For example, when searching for the high-elevation target, the first two periods were characterized by increasingly ambient fixations, then switching to highly focal in the latter two periods of visual search. Meanwhile, in the target-absent condition, although visual search started as focal during the first period, it then switched to ambient, but not until the final epoch. A within-subjects ANOVA of $\mathcal{K}$ over time suggests a statistically significant effect of time ($F(3, 13) = 5.57$, $p < 0.01$, see Fig. 15.6b). with a significant task-time interaction effect ($F(7, 9) = 2.29$, $p < 0.05$).

What is interesting in the analysis of $\mathcal{K}$ dynamics is not only the fact that $\mathcal{K}$ changes over time, but especially the moment when it changes from ambient ($\mathcal{K} < 0$) to focal ($\mathcal{K} > 0$). This may be an indication of visual behavior changing from search to decision, i.e., cognitive examination of whether the fixated spot is the target being sought.

Perhaps somewhat similar to the $\mathcal{K}$ coefficient's discrimination between ambient and focal characteristics of fixations, the Nearest Neighbor Index, or NNI, denoted by symbol $\mathcal{R}$, can provide an indication of fixational spatial grouping. Recall that the NNI is a measure of dispersion, and, as such, suggests that the spatial distribution of data points, e.g., fixations, is either ordered ($\mathcal{R} > 1$), random ($\mathcal{R} = 1$), clustered ($\mathcal{R} < 1$), or maximally aggregated, i.e., singular ($\mathcal{R} = 0$). That is, the smaller $\mathcal{R}$ is, the closer the distribution is towards a singularity (a single point). In the present example, Fig. 15.7a shows that $\mathcal{R}$ decreases with apparent diminished task difficulty. The easier the task, the less dispersed the fixations may become. In this instance, however, the effect of task difficulty on $\mathcal{R}$ is not significant ($F(1, 3) = 2.28$, $p = 0.11$, $n.s.$).



(a) Nearest Neighbor Index.

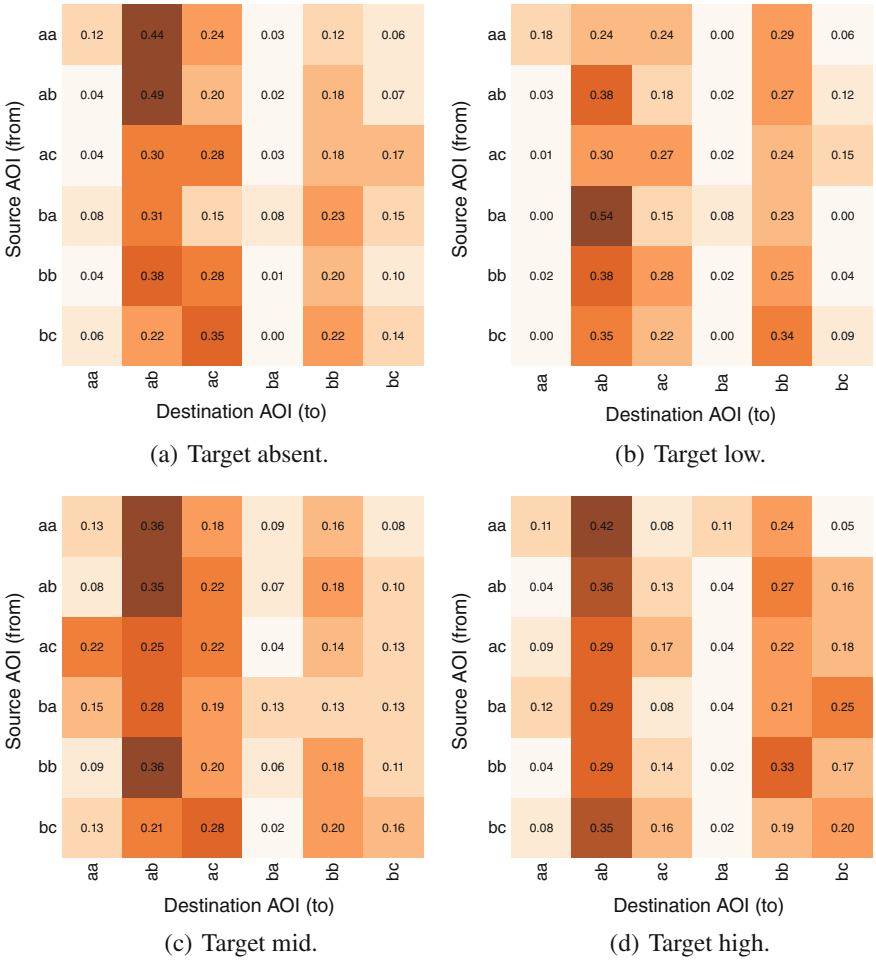
(b) Transition entropy.

Fig. 15.7 Analysis of Nearest Neighbor Index and transition entropy (2 × 3 AOI grid)

Finally, as another type of analysis of visual search dynamics, transition entropy can also provide insight into the nature of the visual task. A within-subjects ANOVA of transition matrix entropy fails to show statistical significance of task ($F(1, 3) = 2.68$, $p = 0.07, n.s.$, see Fig. 15.7b), although it appears that the low-elevation task, thought to be the most difficult of the target-present tasks, is trending toward significant. With transition entropy likened to *surprise*, this analysis suggests lower predictability of where gaze is likely to transition to given its present location.

Consider the transition matrices shown in Fig. 15.8. Transition matrices depict the observed probabilities of transition from one AOI to another. In this instance, a uniform 2 × 3 grid was used to demark AOIs on the stimulus (i.e., 2 columns by 3 rows), resulting in 6 AOIs total. Each transition matrix is therefore a 6 × 6 matrix. The transition matrix corresponding to the low-elevation condition, shown in Fig. 15.8b appears lightest in color, suggesting a more uniform distribution of transition probabilities. Compare and contrast with the high-elevation condition, shown in Fig. 15.8c, which contains a fairly dark band in the second column. This suggests, on average, higher probability of switching to the middle row on the left side of the image relative to any other cell in the grid (see Fig. 15.4).

Transition entropy analysis gives an interesting view of the distribution of visual attention in terms of which grid AOI is likely to be fixated next. Using the surprise analogy, consider the two possible extremes. On the one hand, if the viewer kept fixating a single AOI, there would be no surprise as to what is going to be fixated next, there would be no surprise and 0 entropy. On the other hand, if each time a new AOI was fixated such that eventually every AOI was equally likely, then any AOI would be equally as likely to be selected, hence maximum surprise, or maximum entropy (1 if normalized).



(a) Target absent. (b) Target low.

(c) Target mid. (d) Target high.

Fig. 15.8 Transition matrices

Grid granularity has an impact on transition entropy analysis. If the grid AOIs are too small, then a larger proprtion of AOIs will be less likely to be transitioned to. If the grid AOIs are too large, then transitions between them may become less meaningful. Either way, entropy statistics may become less meaningful. Table 15.1 illustrates the effect of grid granularity on entropy. Finding a grid tesselation that is optimal, e.g., in some statistical or other sense, is an open research problem.

Although grid-based transition entropy is fairly straightforward to implement, transition entropy analysis can also be applied to AOIs that are created based on the underlying visual stimulus. For example, if examining how faces are viewed, AOIs can then be established over the eyes, mouth, and nose regions. This would produce 3 AOIs for which transition entropy may be more meaningful than with a uniform grid.

Table 15.1 Effect of grid size on inferrential statistics

Grid	F-statistic	p-value
1×3	$F(1, 3) = 10.86$	$p < 0.01$
2×1	$F(1, 3) = 3.92$	$p < 0.05$
2×3	$F(1, 3) = 2.68$	$p = 0.07, n.s.$
4×3	$F(1, 3) = 0.74$	$p = 0.54, n.s.$

15.3 Summary and Further Reading

A gaze analytics pipeline, e.g., one that is implemented in `Python`, was described, split into its 5 main steps. The gist of the approach is that raw gaze data are split up into their individual scanpaths, i.e., one per viewing condition (and per participant). Most of the analytical metrics based on eye movements are derived from individual scanpaths, i.e., a sequence of fixations. Metrics such as fixations, fixation durations, etc., are then aggregated for statistical analyses.

Because of the split-then-merge approach, analysis may be time-consuming and prone to generation of a large number of files, starting with the `.raw` files. Processing each `.raw` file in a sequential manner may take some time. It is possible to speed up the process substantially via distributed computing. One such strategy is described by Duchowski (2015), wherein a High Performance Computing (HPC) cluster is used for the purpose.