The Salient360! Toolbox: Processing, Visualising and Comparing Gaze Data in 3D
Erwan David, Jesús Gutiérrez, Melissa Le-Hoa Vo, Antoine Coutrot, Matthieu Perreira Da Silva, Patrick Le Callet

To cite this version:


HAL Id: hal-04139630
https://hal.science/hal-04139630
Submitted on 23 Jun 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
The Salient360! Toolbox: Processing, Visualising and Comparing Gaze Data in 3D

Erwan David
Scene Grammar Lab, Department of Psychology, Goethe University
Frankfurt-am-Main, Germany
david@psych.uni-frankfurt.de

Jesús Gutiérrez
Grupo de Tratamiento de Imágenes, Universidad Politécnica de Madrid
Madrid, Spain
jesus.gutierrez@upm.es

Melissa Lè-Hoa Võ
Scene Grammar Lab, Department of Psychology, Goethe University
Frankfurt-am-Main, Germany
mlvo@psych.uni-frankfurt.de

Antoine Coutrot
LIRIS, CNRS, University of Lyon
Lyon, France
antoine.coutrot@cnrs.fr

Matthieu Perreira Da Silva
Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004
F-44000 Nantes, France
matthieu.perreiradasilva@univ-nantes.fr

Patrick Le Callet
Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004
F-44000 Nantes, France
patrick.le-callet@univ-nantes.fr

ABSTRACT
Eye tracking can serve as a gateway to studying the mind. For this reason it has been adopted by a diverse range of scientific communities. With the improvement of the quality of head-mounted virtual reality devices (HMDs) over the past 10 years, eye tracking has been added to capture gaze in immersive environments. The use of HMDs with eye tracking is increasing significantly and so is the need for a toolbox enabling consensus about eye tracking methods in 3D. We present the Salient360! toolbox: it implements functions to identify saccades and fixations and output gaze characteristics (e.g., fixation duration or saccade directions), to generate saliency maps, fixation maps, and scanpath data. It also implements routines made to compare gaze data that were adapted to 3D. We hope that this toolbox will spark discussions about the methodology of 3D gaze processing, facilitate running experiments, and improve the gaze study in 3D.

https://github.com/David-Ef/salient360Toolbox

CCS CONCEPTS
• Software and its engineering → Software libraries and repositories;
• Human-centered computing → Visualization toolkits;
• Information systems → Extraction, transformation and loading.

KEYWORDS
toolbox, gaze tracking, head tracking, 360 stimuli, processing, comparison, visualisation

1 INTRODUCTION
Gaze tracking is highly utilised throughout vision sciences, it is particularly used as an overt cue for visual attention. Gaze positions and movements are made up of an extremely rich spatio-temporal signal [Liversedge and Findlay 2000], the importance of which has been recognised by a diverse array of scientific and industrial communities. Amongst many other examples, gaze data is used to predict experimental [David et al. 2019; Le Meur et al. 2017] or clinical [Beltrán et al. 2018; Zhang et al. 2016] groups. In clinical settings it has been applied to the development of screening tests [Benfatto et al. 2016; Crabb et al. 2014] and treatment/rehabilitation training [Livengood and Baker 2015]. In training scenarios it provides better feedback for learners [Rosch and Vogel-Walcutt 2013]. It shapes image and video compression algorithms [Dias et al. 2015; Li et al. 2011; Zhang et al. 2021] and image/video quality metrics [Le Meur and Coutrot 2016; Le Meur et al. 2010; Ninassi et al. 2007]. The cinema industry understands its usefulness in the analysis of the locus of attention in narrative contents [Lőwe et al. 2015; Marañes et al. 2020].

The quality of virtual reality (VR) headsets has increased tremendously in the last decades. With the addition of embedded eye trackers, scientists have begun to rely on it more and more to study gaze and visual attention in immersive conditions closer to the natural world. As these devices enter homes, more immersive and 360 contents (image, movies, video games) specific to this viewing paradigm are now being created. As a result, an understanding of how people look and behave in these environments [David et al. 2022; Sitzmann et al. 2018] is required to improve, for example, gaze prediction algorithms, which is essential to developing compression algorithms adapted to VR. Consequently, there is a need to have robust and powerful tools to process gaze, eye, and head tracking in 3D.

There exist several eye tracking toolboxes meant to handle data obtained on standard computer screens. For example, eye tracker...
vendors often provide tools themselves, for example to identify saccades and fixations, and create saliency maps. Some of these toolboxes are dedicated to a particular eye tracker [Cornelissen et al. 2002]. There are non-specialised toolboxes meant to be used in many circumstances [Andreu-Perez et al. 2016; Krassanakis et al. 2014], while others are dedicated to particular analyses [Moacdieh and Sarter 2012], applications [Cercenelli et al. 2017] or experimental conditions [Marighetto et al. 2017].

When transitioning from on-screen to VR studies, it becomes clear that eye tracking toolboxes are not applicable. Moving from a screen as a 2D plane to a 3D world, one must now consider the movement of the head as part of the gaze. Therefore, eye rotation data are often referenced by "eye-in-head" and the combined head and eye rotation by "eye-in-space" [Lappi 2016; Larsson et al. 2016]. In addition to this nomenclature, in this paper we choose to employ eye and gaze, respectively. On top of this, gaze-parsing algorithms need to be modified (e.g., the Euclidean distance used to compute velocities is replaced by the orthodromic distance). Many measures must be updated, which are used to process raw data, generate features of saccades and fixations, or saliency data. Saliency and scanpath comparison algorithms [Bylinskii et al. 2016; Le Meur and Baccino 2013] need modifying as well. The toolbox described in this article was created in response to this void in the community. Its functionalities related to saccade and fixation features are useful to experiments tracking head and eye, whereas those related to saliency maps are constrained to omnidirectional stimuli, like 360° pictures and videos. As more and more teams come to use gaze tracking in VR we believe it is of utmost importance to publish updated tools and start broader discussions about new/adapted tools and methods.

2 WHAT THE SALIENT360! TOOLBOX IMPLEMENTS

The first version of the toolbox was written to handle eye and head rotation data gathered for the Salient360! visual attention modelling challenge [David et al. 2018; Gutiérrez et al. 2018b,a]. The requirements were to:

- Combine eye (eye-in-head) with head rotation data to get gaze (eye-in-space)
- Process head data
- Identify fixations and saccades
- Calculate features of fixations and saccades
- Generate saliency maps and scanpath
- Compare saliency maps and scanpaths

In its latest version the toolbox handles all of this better than it did in 2018. In addition, it now allows to visualise gaze data (raw and processed). The toolbox is written in Python 3 (Python Software Foundation, https://www.python.org/) with the help of the scientific computing toolbox (SciPy [Virtanen et al. 2020], with NumPy [Harris et al. 2020]) and statsmodels [Seabold and Perktold 2010]. The OpenCV [Bradski 2000] and scikit-image [van der Walt et al. 2014] modules are used to manipulate saliency maps as images. Numba [Lam et al. 2015] is used to accelerate some processing steps (e.g., saliency map calculation), PyOpenGL and PyQt5 were needed to build the visualising part of the toolbox. A list of requirements and an installation scripts are provided in the repository’s README file. Installation is made easier with the use of a Conda environment.

It should be noted that the main purpose of the toolbox is to handle gaze data as the combination of eye and head data, but it is useful to work with head data alone as well. Indeed, in the context of the Salient360! challenge it was also used to generate trajectories (akin to gaze scanpath) and saliency maps based on these head trajectory data. So keep in mind that, although we describe most of the toolbox as pertaining to gaze data, it can be used to process head data alone as well.

2.1 Processing

The minimal input is, for each sample: a timestamp, an eye direction vector, and a head rotation value (Euler angles or quaternions). When loading a raw or processed gaze file, the toolbox will try to identify...
the required variables amongst the file columns according to its header and a set of allowed variable names.

Raw data are processed to produce saccade and fixation features according to a set of allowed parameters. First, according to what eye data is available, one may choose which to use: left, right or combined. If combined eye data is not provided it will be computed as the average of left and right data. We recommend to resample head and eye data to have matching sampling rates, for example, using the Vive Pro Eye will result in head data sampled at 90Hz and eye tracking data at 120Hz. Head rotations are stored as quaternions (if provided as Euler angles, they will be converted), thus we use the spherical quadrangle interpolation (SQUAD) method [Shoemake 1985]. Cubic interpolation is used for eye direction vectors. To identify saccades and fixations [Salvucci and Goldberg 2000] we provide three methods:

- A velocity-based method – using a velocity threshold parameter (in °/s).
- A hidden Markov model method – model’s parameters are trained on the velocity signals and hidden states come to represent samples of low (fixations) and high (saccades) velocities.
- A cluster-based method – DBSCAN [Ester et al. 1996] is fed sample positions and used to separate clusters of points (fixations) from noise (saccades).

Filter parameters to smooth the velocity signal can be provided as well (Gaussian or Savitzky–Golay filters).

Head data can be processed alone to produce a head trajectory. In that case we use a sliding time-window (90ms) to calculate the average position of the head, as if the gaze were centred in the visual field of view (forward vector of the head tracking data projected on a unit sphere).

2.2 Scanpath and saliency generation

Once fixation and saccades have been identified, one can produce saliency maps as a general representation of where the gaze landed in the scene or one can produce a time sequence of features that we will call fixation list or scanpath.

Note that we consider gaze data to be projected on a sphere surrounding the observer’s head, to represent this information visually and to be saved on disk this is projected using the equirectangular projection. Therefore, saliency maps are created by drawing a 2D Gaussian at the location of all identified fixations on the equirectangular map. The Gaussian kernels drawn are isotropic on the sphere, but not on the equirectangular map due to the latitudinal distortion resulting from the cylindrical projection. The default standard deviation of the Gaussian kernel is set to two degrees.

Video saliency maps can be generated to support protocols presenting dynamic stimuli, like videos. In that case, a frame index

---

Figure 2: Example outputs from our toolbox. a) a fixation map as a 2D matrix with fixation counts pixel-wise. b) a saliency map obtained by convolving a fixation map with a Gaussian kernel. c) a saliency map blended with an image to better identify salient regions. d) colour-coded points drawn at fixation locations to get a scanpath image showing time-course development (data from only one trial is shown here); lines between points can be added to help visualise transitions.
must be provided along with the raw data, saliency will then be computed frame-wise. A function is provided to output these saliency frames as images, which can be spliced with ffmpeg [Tomar 2006] to produce a saliency video. Blended saliency images and videos are produced if an image or a video is provided. The saliency data is added over the original stimulus with an opacity of 70%.

Scanpaths (i.e., fixation list) are saved as CSV-formatted files containing either one or all 10 calculated features (Tab 1). In addition to these features, we provide an index of the fixation/saccade, as well as the start and end timestamps of fixations. In the case of head trajectory data the exact same set of features will be calculated, considering time-window points as fixations, two such “fixations” make the start and end point of a “saccade”. A simple version of a scanpath can be output as coloured points at the location of fixations over a black background or the original stimulus.

2.3 Comparing
To compare saliency data we rely on the equirectangular saliency maps generated above. The comparison methods provided are AUC (Borji and Judd ), CC, KLD, NSS, and SIM (see [Kümmerer et al. 2015] and [Bylinskii et al. 2016] for a review of these metrics). Implementations of AUC (Borji and Judd), CC, NSS and SIM are originally by Chencan Qian 1. Since an equirectangular projection shows strong latitudinal distortions we added a correction in the form of a weight vector, in order to give less importance to points near the poles (sine function). This correction is applied for CC, KLD, and SIM. PyTorch [Paszke et al. 2017] implementations of CC, KLD, NSS and SIM measures are provided in the interest of performance and compatibility, though the toolbox will use non-PyTorch implementations by default which are accelerated with Numba.

Scanpaths (time series of saccade/fixation features) are compared using the MultiMatch method [Dewhurst et al. 2012]. This method does not rely on regions of interest, rather it tries to compare the shape of the scanpaths. It considers and reports several measures of scanpaths:

- Direction – Difference between saccade relative angles [David et al. 2022].
- Duration – Difference between fixation durations.
- Length – Difference between saccade lengths.
- Position – Angular distance between fixations on sphere.
- Shape – Difference between saccade “shapes” ($s_{1}c_{1} - s_{2}c_{2}$).

Note that the measures are normalised between 0 and 1, allowing to be interpreted as a percentage of dissimilarity and to be averaged together to produce a general single-value dissimilarity score. Thus, all measures are divided by pi, apart from the duration metric which is divided by the maximum duration observed in the two scanpaths compared.

When comparing dynamic saliency maps or scanpaths, data is compared over sequential windows of approximately 2sec. For every time window we calculate measures, then we average the results over the time windows.

2.4 Visualising
The graphical interface allows to visually assess the quality of the data, to experiment with parameters related to the processing and generating functions (Fig 3). It implements exporting functionalities to output saliency maps (it does not support dynamic stimuli), scanpath as coloured points on the original stimulus or fixation lists with the features of your choice. The GUI offers an equirectangular view window on which gaze data appears as points (raw gaze sample or fixation position) over an image, a greyscale saliency map or an image blending the stimulus with the saliency data (Fig 4). This view also presents a spheres on which the equirectangular data is back-projected (lower-right), along with a viewport (lower-right) approximating what would be perceived in a VR headset. The viewport’s boundary appears on the equirectangular map as a deformed square with a red border.

If a CSV file is dropped over the equirectangular view, the toolbox will check if it contains raw gaze data or if it is a fixation list. The content will then be processed to produce saliency maps and (raw and fixation list) scanpath data. If the Control key is pressed while dropping a text file, its data will be added to what was currently on screen instead of replacing it.

3 USAGES
One can use the toolbox one of three ways, with the:
- Scripting interface (Python)
- Command line interface (CLI)
- Graphical user interface (GUI)

The scripting interface is the most complete and the one we recommend for processing entire databases. It contains a “helper” module (helper.py) meant to simplify accessing particular procedure. For example, getting features of saccades and fixations from raw data requires parsing a file, processing raw data, labelling samples as fixation or saccades, then calculating features. This is streamlined as the getFixationList function which takes as input a path to a CSV file along with the parameters related to all the steps involved. The function will deduce the variables it needs from the CSV file’s header. Similarly, the getSaliencyMap functions takes as input a fixation list (part of the output from getFixationList) and returns a saliency maps, along the way it handles static and dynamic saliency generation, and caching the raw saliency data.

Below is an example of reading, processing, and generating outputs:

```python
# Tracking can be HE (Head+Eye) or H (Head alone)
tracking = "HE"
# Targeted eye
eye = "R"
# Resampling rate
resample = 120
# Filter settings
filterSettings = {"name": "savgol", "params": {"win": 9, "poly": 2}}
# Gaze parsing settings
parsingSettings = {"name": "I-VT", "params": {"threshold": 120}}
# Dimensions of output images (Height, Width)
```

---

### Table 1: List of saccade and fixation features calculated by the toolbox, and available to be saved in CVS files.

<table>
<thead>
<tr>
<th>Event</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixation</td>
<td>Duration</td>
<td>Time difference between first and last sample making up the fixation</td>
</tr>
<tr>
<td></td>
<td>Position</td>
<td>Average gaze or head position as longitudes and latitudes or as unit direction vectors</td>
</tr>
<tr>
<td></td>
<td>Dispersion</td>
<td>Average distance of samples’ position making up the fixation and its centroid</td>
</tr>
<tr>
<td></td>
<td>Peak velocity</td>
<td>Maximum velocity observed during the fixation</td>
</tr>
<tr>
<td></td>
<td>Peak acceleration</td>
<td>Maximum acceleration observed during the fixation</td>
</tr>
<tr>
<td>Saccade</td>
<td>Amplitude</td>
<td>Angular distance between first and last saccade sample on the sphere</td>
</tr>
<tr>
<td></td>
<td>Absolute angle</td>
<td>Angle between the saccade vector and the longitudinal axis</td>
</tr>
<tr>
<td></td>
<td>Relative angle</td>
<td>Angle between two consecutive saccade vectors</td>
</tr>
<tr>
<td></td>
<td>Peak velocity</td>
<td>Maximum velocity observed during the saccade</td>
</tr>
<tr>
<td></td>
<td>Peak acceleration</td>
<td>Maximum acceleration observed during the saccade</td>
</tr>
</tbody>
</table>

---

**Figure 3:** Screenshot of the options offered by the GUI.

```python
# Example code block for loading and processing data

dim = [500, 1000]  # Path to CSV file containing raw gaze data
path_raw_file = "/path/to/file.csv"

path_stim = "/path/to/stim.png"

from Salient360Toolbox import helper

# Get processed raw data and list of fix/sacc features
gaze_data, fix_list = helper.loadRawData(path_raw_file,
    eye=eye,  # If gaze tracking, which eye to extract
    tracking=tracking,  # Gaze or Head tracking
    resample=resample,  # Resampling at a different sample rate?
    filter=filterSettings,  # Filtering algo and parameters if any is selected
    parser=parsingSettings,  # Fixation identifier algo and its parameters
)

# Generate saliency map from loaded data
sal_map = helper.getSaliencyMap(fix_list[:, [2,3,4, 0,1]],
    dim,  # Name of binary saliency file created for caching purposes
    name=savename,
)

# If a binary file exists at this location we load the saliency data from it, unless force_generate is True. Saliency will be saved if caching is True
path_save=PATH_OUT,

# Sigma of the 2D Gaussian drawn at the location of fixations
# Asks to return saliency data rather than a path to a saliency data file if it exists
force_return_data=True,

force_generate=False,  # Will save saliency to binary file to fast load at a later time

caching=True

# Get a fixation map (2d matrix with number of fixations observed at each pixel location)
fix_map = helper.getFixationMap(fix_list[:, :2], dim)

from Salient360Toolbox.generation import saliency

sal_image = saliency.toImage(sal_map, cmap="coolwarm")
```
Figure 4: Screenshot of the equirectangular rendering in the GUI with a sphere (lower-left) and a viewport (lower-right) showing other projections of the data.

```python
# (fig 2.a) Save fixation map as a gray scale image
fix_map_img = sal_generate.toImage(fix_map, cmap="binary",
reverse=True, blend=path_stim)
sal_generate.saveImage(fix_map_img, outpath="_fixmap")

# (fig 2.b) Save saliency map as greyscale image
sal_generate.saveImage(sal_image, outpath="_salmap")

# (fig 2.c) Save saliency map blended with stimulus
sal_generate.saveImage(sal_map, outpath="_bsalmap",
blend=path_stim)

# (fig 2.d) Save stimulus with fixation points drawn over
# it
scanp_generate.toImage(fix_list[:, :, :], dim,
outpath="_bscanpath", blend=path_stim)

from Salient360Toolbox.generation import scanpath as scanp_generate

# Save scanpath data (fixation and saccade features) to
# file
scanp_generate.toFile(fix_list, outpath="_fixation.csv",
# Save all features
saveArr=np.arange(fix_list.shape[1]), mode="w")
```

The CLI is handy to access the toolbox through the shell or invoked via another process. It exposes all functionalities of the toolbox.

The GUI is the most limited of the three solutions because it does not allow batch processing of files in order to automatically produce outputs for a database. Data is added by drag-and-dropping files onto the window: an image will be used to set the background; a video will be used to set the background (a frame will be extracted); a CSV file will replace or update gaze data on-screen.

4 CONCLUSION

The advent of virtual reality devices with embedded eye trackers allows for easy tracking of head and eye rotations to study eye-in-space behaviour in immersive and controlled conditions. Although, the study of eye movement on 2D screens has produced strong methods, they are not necessarily directly transferable to 3D gaze data. The Salient360! Toolbox provides functionalities for extracting, comparing and visualising gaze data obtained in 3D contexts (such as with a VR headset), where head and eye rotations were measured. Note that its functionalities related to saliency map calculation and comparison are applicable particularly to omnidirectional stimuli (360 images or videos). Our toolbox constitutes efforts meant to
democratise the use of head and eye tracking devices by making it easier to use the resulting data. With the release of this tool-box we anticipate discussions about best practices and methods that will certainly lead to improvements and consensus within the communities relying on head and eye tracking.

ACKNOWLEDGMENTS

This work supported by RFI Atlantic2020, the SFB/TRR 26 135 project C7 to Melissa L.-H. Võ and the Hessisches Ministerium für Wissenschaft und Kunst (HMWK; project 'The Adaptive Mind').

REFERENCES


