IGazeLib Documentation

Contents

[Introduction 3](#_Toc356207746)

[Data structures 4](#_Toc356207747)

[Functions 5](#_Toc356207748)

# Introduction

IGazeLib is a library that aims to facilitate the analysis of eyetracking- or gazedata. The library consists of a group of eyetracking functions from low-level to high level. This document serves as a manual to provide easy access to the usage of the toolbox.

Library gives the user powerful tools to perform offline data analysis on eyetracking-data. The user can write a script file aided by the library functions to perform various analyses

The library has been tested and works on Matlab R2012b on Os X 10.6.8 and Windows 7. However, we cannot see why it would not work on other versions too. Some of the functions may work on Octave, but may also contain parts that are not compatible, e.g. the containers.Map-datatype.

Installation instructions please

# Data structures

The central data structure of the program is the gaze data-matrix. The raw eye-tracking data is usually interpreted in the form of rows representing data points and columns representing various metrics of each data point. This library assumes that the data is presented in the form of cell vector describing n column headers and a data matrix with n columns and m rows. The amount of rows or columns is not limited in any way, except memory.

* HEADERS is a cell-array containing strings.
* DATA is a cell-array containing vectors of different type, e.g. strings, integers or floats.

Overview of an example of the central data structure:

HEADERS: cell vector with the values

‘ID’, ‘TETTime’, ‘ValidityLeftEye’, ‘ValidityRightEye’, ’XGazePosLeftEye’, ‘XGazePosRightEye’, ‘YGazePosLeftEye’, ‘YGazePosRightEye’, ‘Stim’

DATA: cell-vector containing 9 vectors format: int, float, float, float, float, int, float, float, int, int, cell (here represented as a csv-array)

1, 1.34E+12, 0.545283, 0.481257, 0.609004, 0, 0.466017, 0.520776, 0, 1, control.bmp

2, 1.34E+12, 0.532888, 0.48524, 0.609017, 0, 0.451526, 0.496512, 0, 1, control.bmp

3, 1.34E+12, 0.539892, 0.476169, 0.609034, 0, 0.465413, 0.500851, 0, 1, control.bmp

4, 1.34E+12, 0.536817, 0.512887, 0.609045, 0, 0.458263, 0.475376, 0, 1, control.bmp

5, 1.34E+12, 0.544735, 0.499023, 0.609053, 0, 0.47424, 0.503782, 0, 1, control.bmp

6, 1.34E+12, 0.531206, 0.42684, 0.609036, 2, 0.451224, 0.501626, 2, 1, control.bmp

7, 1.34E+12, 0.545374, 0.496693, 0.609034, 2, 0.458163, 0.562241, 2, 1, control.bmp

8, 1.34E+12, 0.531187, 0.479646, 0.609101, 0, 0.497042, 0.4768, 0, 1, control.bmp

# Function types

Functions include various types of functions. The functions can be categorized as follows:

* Loading functions – functions that process raw data and load that in the appropriate form to the data structure.
* Clipping functions – a group of functions designed to clip the data accordingly to fit the analysis period.
* Preprocessing functions – a group of functions to perform preprocessing tasks, e.g. interpolation.
* Calculation-functions – functions that calculate various metrics from the data
* Visualization functions – a group of functions to perform visualization of the data
* Data saving functions
* Analysis scripts – scripts that are combined from these building blocks to perform a certain type of analysis.
* Other functions – various functions to help performing certain tasks

# Building an analysis

Normally, a good way to implement an analysis is to first cut or segment the data as needed and then calculate metrics or perform a visualization.

### Example analysis

This is an example of how to use the library to perform a simple analysis. Here we clip the data according to the column ‘Stim’ and calculate the time when gaze leaves our aoi. We also want to see the visualization of each gaze when the analysis is running and combine the results in a nice format.

Pseudo code:

1. Read files
2. Clip the data so, that each time there is a change in column ‘Stim’, form a new clip
3. Do for each clip
   1. Calculate when gaze leaves our aoi
   2. Perform visualization
4. Save data in a csv-file

Matlab code:

AOI = [ 0.2 0.2 0.8 0.8];

[DATA, HEADERS] = loadGazeFile(filename);

clipped\_data = clipDataWhenChangeInCol(DATA, findColumnNumber(HEADERS, ‘Stim’));

for i=1:length(clipped\_data)

stim{i} = getValue(clipped\_data, 1, findColumnNumber(‘Stim’));

gazerow = gazeInAOI(clipped\_data{i}, HEADERS, AOI, ‘last’);

t\_start = getValue(clipped\_data, 1, findColumnNumber(‘TETTime’));

t\_dis = getValue(clipped\_data, gazerow, findColumnNumber(‘TETTime’));

offset(i) = t\_dis - t\_start;

plotGazeAnimation(clipped\_data{i});

end

saveCsvFile({‘Stimulus’, ‘Last time in aoi’}, stim, offset);

# Notes

It is of responsibility of user to assure that the calibration is correct and display limits are similar to the physical dimensions of the eyetracker screen.

The user is responsible for all uses/causes of using this library.

# Function listing

Fresh function listing can be printed from the Matlab by calling function

printFunctionOverview()

Function [gaze\_violation] = AOIBorderViolationDuringNonValidSection(DATA, HEADERS, aoicoord, accepted\_validities)

Returns a truth value if the gaze moves over the aoi border during

invalid data.

Function [DATA, HEADERS] = addNewColumn(DATA, HEADERS, newcolumn, newcolumntitle):

Function adds new column to the DATA matrix and HEADERS-cell vector.

After this operation the column can be operated as it was always part of

.gazedata.

Function [DATA] = clipDataWhenChangeInCol(uDATA, column)

Clips uDATA and returns DATA, a clipped cell-table. Data is being clipped

according to the column (number) when the value in the column is

different from previous row a new clip is formed. This function can

handle both a numerical and cellstring column values as separators.

Function [DATA] = clipFirstMilliSeconds(DATA, HEADERS, millisec)

Returns the first millisec milliseconds of DATA in same format.

Function [DATA] = clipFirstRows(DATA, numrows)

Returns the first numrows rows of DATA in same format.

Function [DATA] = clipLastRows(DATA, cutrow)

Returns the last numrows rows of DATA in same format.

Function [colcount] = columnCount(DATA)

Returns the count of columns in DATA

Function disengagementAnalysis

Analyze file for disengagement-paradigm. Parameters are defined at the

beginning of the script.

Function distanceAnalysis

Analyze file for disengagement-paradigm. Parameters are defined at the

beginning of the script.

Function [distance\_right, distance\_left] = distanceTravelled(DATA, HEADERS, display\_ratio, display\_height)

Calculates the distance moved by eyes during the DATA-section

and returns individual values for each eye. display ratio is a number

that describes how many times wider display is than tall. E.G for 16:10

display, use 1.6. display\_height is the height of the display in SI unit,

meters. E.G. for a 51cm tall display, use 0.51.

Function [column\_number] = findColumnNumber(HEADERS, column\_id)

Finds the order number of the column\_id and returns it as an integer. If

column of that name not found, returns -1.

Function [files] = findGazeFilesInFolder(folder)

Returns the full pathnames of gazefiles in the folder. The gazefiles are

assumed to have a .gazedata-ending.

Function [percentage] = gazeInAOIPercentage(DATA, HEADERS, aoicoord)

Returns the percentage of time when gaze is inside the area aoi. If the

user does not enter aoi, then -1 is returned. Gaze

aoicoord = [xmin xmax ymin ymax] specified in the x and y pixel

coordinates that are beign used (likely ranging from 0..1).

Function [rownumber] = gazeInAOIRow(DATA, HEADERS, aoicoord, firstorlast)

Returns the rownumber when gaze is (according to last/first) inside the area aoi.

If the user does not enter aoi, then -1 is returned. If user does not leave

(or enter) the aoi, -1 is returned.

aoicoord = [xmin xmax ymin ymax] specified in the x and y pixel

coordinates

firstorlast = returns either the first time value when gaze is in the aoi

or last

Function [columnvector] = getColumn(DATA, columnnumber)

Returns the specified column from the DATA-matrix as a vector or

cell-vector according to the data type of the column.

Function [DATA] = getRowsContainingValue(DATA, column, value)

Takes the rows that contain specified value in specified column. Value

can be a value, or a vector of values (strings: cell vector) of values.

Function [value] = getValue(DATA, row, column)

Returns value specified by row and column from the array DATA.

Function [DATA] = interpolateUsingLastGoodValue(DATA, column, validitycolumn, accepted\_validities)

Interpolates values in column "column" in DATA-matrix by replacing the bad

value with last good value before bad values (if there is at least one good

value, otherwise, do nothing). Validitycolumn contains the validity markings

for each datapoint and good validities are defined by the accepted

validities-parameter.

Function [DATA, HEADERS] = loadGazeFile(file)

Loads the gaze file and returns all datalines and columns in DATA-array and

headers in HEADERS-cell-vector. Load function is sensitive in content

format and if the original files differ from the format displayed here, a

new or modified load-function is needed.

Function [DATA, HEADERS] = loadGazeFileBoston(file)

Loads the gaze file and returns all datalines and columns in

DATA-array and headers in HEADERS-cell-vector.

Function [DATA, HEADERS] = loadMatlabGazeFile(file)

Loads the gaze file and returns all datalines and columns in

DATA-array and headers in HEADERS-cell-vector.

Function [longestr, longestl] = longestNonValidSection(data, HEADERS, accepted\_validities)

Calculates the validities for each eye separately. The result is calculated

and returned in TETTime.

Function [DATA] = medianFilterData(DATA, winlen, column)

Performs median filtering to the datapoints with window-length winlen.

Winlen must be an odd integer where the window is equal length ((winlen-1)/2)

in both directions centering in datapoint.

Here datapoints must contain numbers, otherwise an error is presented.

Column specifies the column to filter the data with

Function [max\_moves\_r, max\_moves\_l] = movementSpeedDuringNonValidSection(DATA, HEADERS)

Calculates the speed of movement during non-valid section. Function returns

the maximum speed determined during the DATA. If no interval is present,

a -1 is returned. NVS stands for NonValidSection.

Function [vector] = normalizeVector(vector)

Takes away the mean of the vector.

This function is specifically made to parse the information about the

gaze-timeseries variables for Matlab Psychtoolbox to disengagement

paradigm.

Function plotDetailedGazeAnimation(DATA, HEADERS, figtitle, delaytime, accepted\_validities, savegaze, varargin)

Function plots gaze animation of the participant read from the

DATA-matrix. figtitle specifies the name of the figure. Varargin may

contain still images that are placed in the animation accordingly.

Delaytime is to tune the loop slower if the computer is "too" fast.

varargin format is what follows:

{imagefile, coordinates, startrow, endrow}

coordinates are in the form [xstart xend ystart yend]

Function plotEyeDistAnimation(DATA, HEADERS, figtitle, delaytime, accepted\_validities, varargin)

Function plots the distance of the eyes of the participant read from the

DATA-matrix. figtitle specifies the name of the figure. Varargin may

contain still images that are placed in the animation accordingly.

Delaytime is to tune the loop slower if the computer is "too" fast.

varargin format is what follows:

{imagefile, coordinates, startrow, endrow}

coordinates in the form [xstart xend ystart yend]

Function [hfig] = plotGaze1d(DATA, HEADERS)

Plots the gaze in DATA one-dimensionally. A handle to figure is returned.

Function [hfig] = plotGaze2d(DATA, HEADERS)

Plots the gaze in the DATA-array 2-dimensionally. The figure handle hfig

is returned.

Function plotGazeAnimation(DATA, HEADERS, figtitle, varargin)

Plots the gaze of the participant read from the DATA-matrix. figtitle

specifies the name of the figure. Varargin may contain still images that

are placed in the animation accordingly. varargin format is what follows:

{imagefile, coordinates, startrow, endrow}

Function printFunctionOverview

Prints the overview of functions contained in the toolbox by displaying

their help-texts.

Function [DATA] = removeRowsContainingValue(DATA, column, value)

Takes the rows that do not contain the specified value in specified

column. The rows containing the specified value are discarded.

Function [DATA] = replaceStringsInColumn(DATA, column, value\_before, value\_after)

Replaces all the value\_before's with value\_after in the column specified

by column (number) in DATA.

Function [rowcount] = rowCount(DATA)

Function calculates how many samples (rows) DATA-matrix has in total

(according to the first column).

Function saveCsvFile(filename, headers, varargin)

Function saves the data to a csv-file. Varargin must contain as many

columns as headers contains column headers. A new function is made

because Matlab writecsv contains some problematic aspects, e.g. in

systems with different separator-symbol.

Function saveDetailedGaze(DATA, HEADERS, figtitle, delaytime, accepted\_validities, varargin)

Function plots gaze like the DetailedGazeAnimation and saves the gaze to

a file specified by figtitle.

Function [isgood] = testDataConsistency(DATA)

This function performs tests to see if the data is in right form (e.g.

all the columns have same amount of rows). Additional tests may be

included later on.

Function testScript(file)

Common testing place for new scripts. By default contains simple example

script.

Function [unique\_values] = uniqueColumnValues(DATA, column)

Function returns all the values that are present in the specified column.

For numbers, returns a vector and for strings, returns a cell-vector.

Function [rpercentage, lpercentage] = validGazePercentage(DATA, HEADERS, accepted\_validities)

Function examines data and returns the amount of valid data. Accepted

validities contains the validity marks to be considered "good".