edfread Manual

Reading Eyelink Datafiles into Matlab

Johannes Steger

jss@coders.de

edfread Manual: Reading Eyelink Datafiles into Matlab

by Johannes Steger

Table of Contents

List of Tables

Chapter 1. Introduction

Edfread is a Matlab extension that reads binary "Eyetracker Data Files" (edf for short) that are created by SR-Research's Eyelink system into Matlab variables. It allows for extracting trial samples, fixations, button events, saccades, calibration and drift correction data as well as custom filtering of data messages.

This manual describes usage and structure of edfread and is divided into three parts:

Tutorial

For first-time use in Matlab: How to install and access data.

Matlab Reference

Detailed listing of all available structures and their meaning.

Developer Documentation

Short description of how edfread works and how you can modify it yourself.

Chapter 2. Tutorial

2.1. Installation

Edfread comes in a single folder for all supported platforms and architectures (Windows 2000, Windows XP 32bit, Mac OS X, Linux i386). It comes in two flavors: the "normal" distribution is optimized for current hardware, whereas the "compat" distribution is slower, but compatible with older (pre-Pentium 4) hardware.

To install edfread, carry out the following steps:

i. Checkout the latest Version from SVN:

```
svn co svn+ssh://yourname@hal.ikw.uos.de/srv/svnroot/nbp/eyetracking/edfread/current \
            edfread
```

(in one line, without the backslash) This will create a new directory called `edfread`. If you did this before, just change to that directory and issue:

```
svn up
```

ii. Start Matlab

iii. Add the folder from step (i) to Matlab's path via File -> Set Path -> Add Folder -> Save

iv. Type "help edfread" to view edfread's inbuilt help. If this does not work, re-check your path settings and verify that your architecture and platform are supported (see above).

Upgrading: To upgrade from a earlier version, just overwrite the existing edfread directory with the new version. Also make sure to look at the `CHANGELOG` file, to see if any changes relevant to your data were made.

2.2. Getting started

Edfread takes at least one parameter: the `.edf` file you want to read. Depending on your operating system, the filename might be case-sensitive. In any case, use an absolute path. A call returns up to two structures: The first, trials, contains an entry for each trial in the experiment. The second, meta, contains information about the experiment and calibration data. If you assign the result of edfread to only one variable, you will only get the trial data.

In the following example, trials has the same value after both calls:

Example 2-1. Invocation

```
>> trials = edfread('test.edf');

>> [trials, meta] = edfread('test.edf');

>> trials

trials =

1x280 struct array with fields:
    left
    right
    button

>> meta

meta =
    header: [1x249 char]
    calib: [9x1 struct]
```

2.3. Accessing single trials

Each member of the trials structure represents one trial. It has separate data fields for the left and right eye as well as for button events:

```
>> [trials, meta] = edfread('test.edf');
>> trials(1)

ans =

      left: [1x1 struct]
     right: [1x1 struct]
    button: [1x1 struct]
```

Furthemore, all your randomization data is also put in there. For each Key, a new field will be added that contains your meta data for each trial. For each eye, event data is available in four more sub-structures. Furthermore, drift correction results are stored as [Screen Offset X, Screen Offset Y, Absolute Angle]:

```
>> trials(1).left

ans =

    fixation: [1x1 struct]
     saccade: [1x1 struct]
       blink: [1x1 struct]
     samples: [1x1 struct]
       drift: [17.6000 1.5000 0.5300]
```

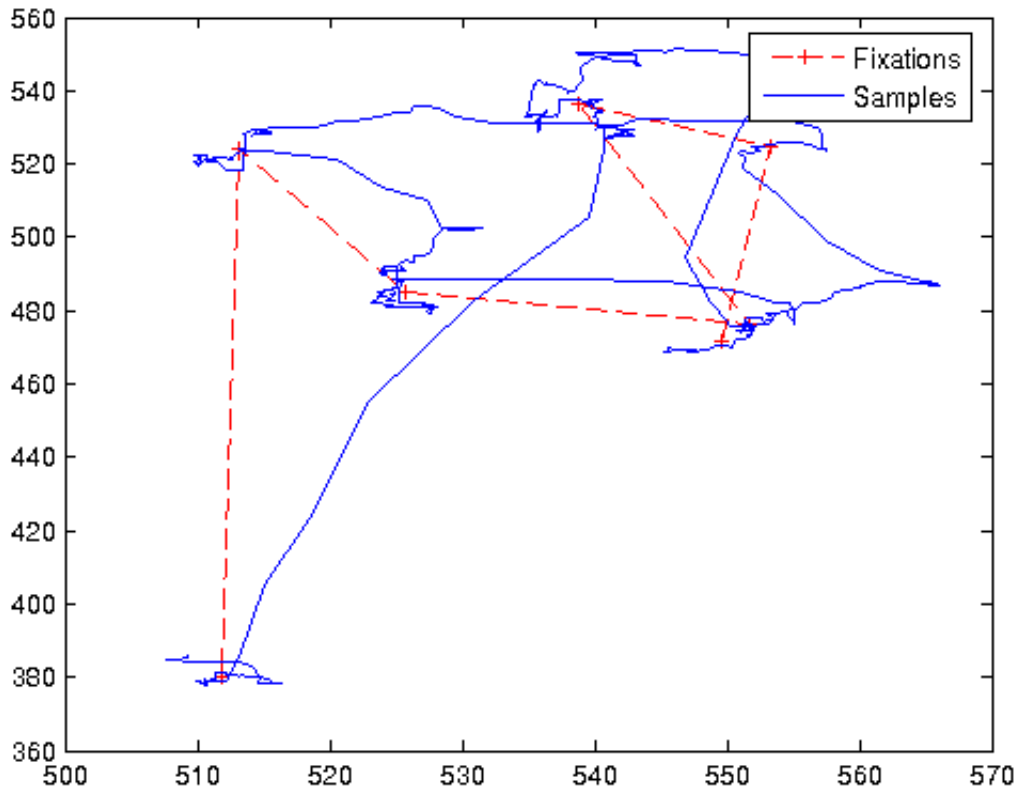
Each event structure contains a variable number of 1xN matrices, where one column is one event:

```
trials(1).left.fixation

ans =

    start: [4 396 796 1412 1580 1924 2352 2512 2984]
      end: [372 764 1392 1544 1884 2248 2492 2968 3180]
        x: [505.2702 442.7034 523.5582 513.3559 450.1506 .. ]
        y: [391.1058 417.8812 561.4643 523.3752 402.1353 .. ]
    pupil: [900.0395 820.1024 1001.2383 956.2332 899.1932 .. ]
```

So to plot all samples and fixations of the left eye for the 10th trial, you would do:

Example 2-2. Plot samples and fixations of trial 10

```
>> plot(trials(10).left.fixation.x,trials(10).left.fixation.y,'+r--')
>> hold on
>> plot(trials(10).left.samples.x,trials(10).left.samples.y,'b-')
>> legend('Fixations', 'Samples')
```



2.4. Accessing global experiment data

The calibration data is stored in the meta-data part of your result:

```
>> [trials, meta] = edfread('test.edf');
>> meta.calib

ans =

9x1 struct array with fields:
    trial
    left
    right
```

Trial is a 1xN integer matrix listing the trials in which calibration took place. Therefore, the calibration at the beginning of the experiment has index 0. Left and right list the results of the corresponding calibration run separately for each eye:

```
>> meta.calib(2).trial

ans =

     0

>> meta.calib(2).left

ans =

    err_avg: 1.1800
    err_max: 2.2600
    off_deg: 3.0800
      off_x: -7.1000
      off_y: -105.2000
      res_x: 0.2000
      res_y: 0
       type: 'HV9'
      coeff: [-1.6402e+03 110.6500 60.9400 -0.0804 ..]
```

Unsuccessful calibration is indicated by error values of NaN and type 'ERR':

```
>> meta.calib(1).left

ans =

    err_avg: NaN
    err_max: NaN
    off_deg: NaN
      off_x: NaN
      off_y: NaN
      res_x: 0.4000
      res_y: 0.1000
       type: 'ERR'
      coeff: [-1.4184e+03 107.1600 64.6780 -0.1811 -4.6111 -2.0804e+03 -13.9520 421.9400 -0
```

Besides the calibration data, global experiment meta data, such as SUBJECTINDEX or EXPERIMENTOR, is also saved to the info structure.

2.5. Combining trials

Often you will want to work on data from several trials (e.g. one condition). The structure of trials does not allow for direct access of more than one trial. However, this is easily overcome by using Matlab's cat or [] operator. Generally speaking, it combines any number of input data sets into a single matrix. So to merge all data from the left eye into left, do:

```
>> left = [trials.left];
>> left

left =

1x280 struct array with fields:
    fixation
    saccade
    blink
    samples
    drift
```

By doing that once again on the fixations, you get an array lfix that contains all fixations of the left eye:

```
>> lfix = [left.fixation];
>> lfix

lfix =

1x280 struct array with fields:
    start
    end
    x
    y
    pupil
```

The reason we decided for this kind of structure is basically better readability: by explicitly selecting the data by their name in the structure, we prevent errors occurring when relying on indexing by number instead. Furthermore, the cat operator is extremely fast - it does not store data, but just links to where it got it from until you actually change it.

Matlab's cat: Sometimes Matlab returns more than one result (i.e. more than one ans) for a given command:

```
>> left(1:2).blink

ans =

    start: 2292
      end: 2312


ans =
```

```
    start: [1752 3044]
      end: [1796 3056]
```

You can cat them together by applying one of:

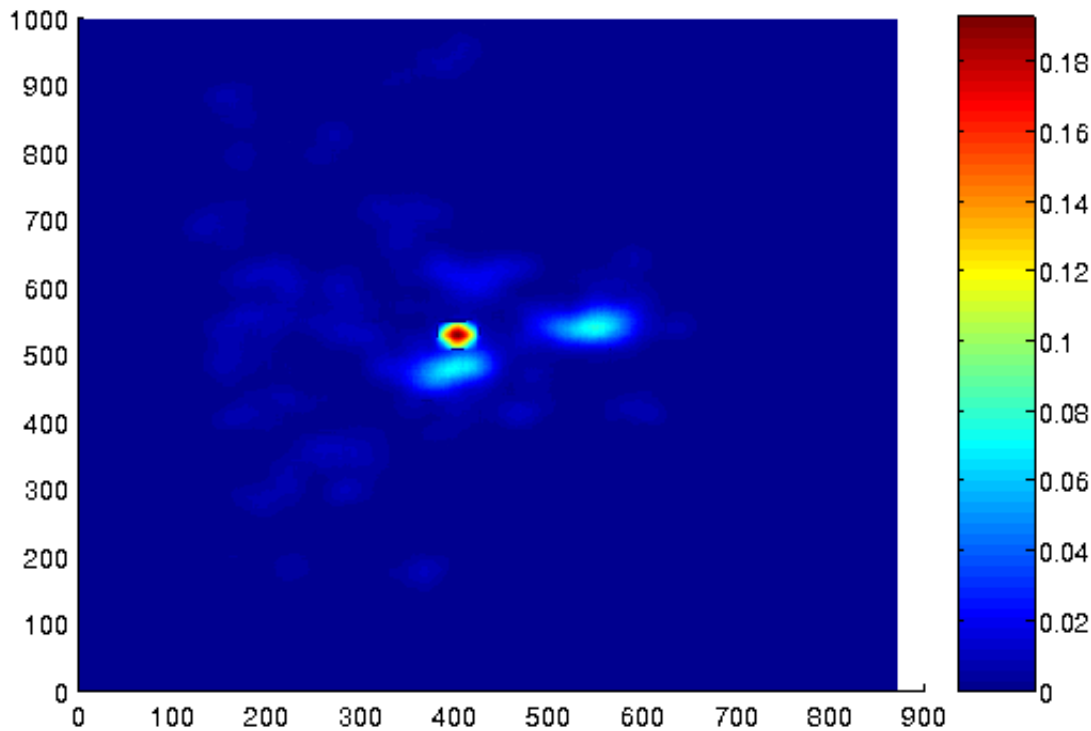
```
>> horzcat(left(1:2).blink)

ans =

1x2 struct array with fields:
    start
    end

>> [left(1:2).blink]

ans =

1x2 struct array with fields:
    start
    end

>> vertcat(left(1:2).blink)

ans =

2x1 struct array with fields:
    start
    end
```

So vertcat(A) == [A]' and horzcat(A) == [A]

Example 2-3. Creating a simple PDF map

```
>> lfix = [left.fixation]';
>> lfixall = [[lfix.x]; [lfix.y]];
>> gauss = meshgrid(gausswin(40)) + meshgrid(gausswin(40))';
>> pdf = accumarray(double(round(lfixall)), 1) / size(lfixall,1);
>> surface(conv2(gauss, pdf), 'EdgeColor', 'none')
```

If you don't know some of these Matlab functions, use the help command to check the documentation.



2.6. Filtering for custom messages

Many experiments include their own messages in the `edf` file. Keyboard events (as opposed to button events) and special condition markers are some of them. Edfread has a generic interface to deal with all of them. You can tell edfread to filter for all messages with a given string (keyword) in it:

```
>> [trials, meta] = edfread('test.edf', 'VFILL', 'KEY');
```

For each keyword, you will find an additional entry in the trial structure:

```
>> trials

trials =

1x280 struct array with fields:
    left
    right
```

```
    button
    VFILL
    KEY
```

Each entry is a structure with two matrices: One, time, contains the timestamps (as always in [ms]) from when the messages were picked up, while the other, msg, contains the message itself as a string:

```
>> trials(1).KEY

ans =

    time: 5029
     msg: 'KEY 4'
```

Chapter 3. Matlab Reference

3.1. Trial Data

3.1.1. Eyes: left and right

Times are indexed in ms from start of the experiment. More information about the contained data can be found in the "Eyelink II User Manual" in Section 4.9.

Table 3-1. fixation

Name	Type	Value	Unit
start	1xN int	Start time of the fixation	ms
end	1xN int	End time of the fixation	ms
x	1xN single	x position	pixels
y	1xN single	y position	pixels
pupil	1xN single	pupil size	

Calculation of fixation coordinates: For the x and y coordinates, all sampled positions between fixation start and stop are taken into account and weighted using a triangular function. The same is done for the pupil size.

Table 3-2. saccade

Name	Type	Value	Unit
start	1xN int	Start time of saccade	ms
sx	1xN single	Start position x	pixels
sy	1xN single	Start position y	pixels
end	1xN int	End time of saccade	ms
ex	1xN single	End position x	pixels
ey	1xN single	End position y	pixels
speed	1xN single	Peek velocity	degrees/s

Table 3-3. blink

Name	Type	Value	Unit
start	1xN int	Start of an eye blink	ms

Name	Type	Value	Unit
end	1xN int	End of eye blink	ms

Table 3-4. samples

Name	Type	Value	Unit
time	1xN int	Time of sample	ms
x	1xN single	Raw sample x position	pixels
y	1xN single	Raw sample y position	pixels
pupil	1xN single	Pupil size	mm diameter

Samples are the raw data recorded by the tracker. Depending on your recording frequency, you will get one sample every 2 or 4ms.

Table 3-5. drift

Name	Type	Value	Unit
1	double	Screen offset X	pixels
2	double	Screen offset Y	pixels
3	double	Absolute offset angle	degrees

Dirft Correction values of "NaN" indicate that it was skipped during this trial.

3.1.2. Buttons

Table 3-6. blink

Name	Type	Value	Unit
time	1xN int	Timestamp of the event	ms
code	1xN int	Button code	id

> **Button vs. Keypress:** "Button press" refers to usage of the game controller attached directly to the host PC. This is the only input you can read by default. If you use the PC keyboard attached to the display PC, you need to manually extract the events using the filter functionality described earlier in Chapter 2 - Tutorial.

3.2. Meta Information

3.2.1. Calibration Data: calib

3.2.1.1. Trial

A simple 1xN matrix stating the trials in which calibration procedures were run. A value of 0 denotes the pre-experiment calibration. There may be more than one entry with the same trial id when several calibration runs were necessary to reach acceptable error values.

3.2.1.2. Left, Right

Table 3-7. calib

Name	Type	Value	Unit
err_avg	single	Average error over all calibration points	degrees
err_max	single	Maximum error	degrees
off_deg	single	Offset	degrees
off_x	single	X offset	pixels
off_y	single	Y offset	pixels
res_x	single	X resolution	degrees
res_y	single	Y resolution	degrees
Type	string	Calibration type: HV9 for nine point	
coeff	1x10 single	Internal Eyelink: Coefficients for the least square fit: X=a+bx+cy+dxx+eyy, Y=f+gx+goaly+ixx+jyy	-

Chapter 4. Utility Functions

After initially reading the data with edfread, you will do further processing on it. Most shared functions in the lab will accept a data format called fixmat. We provide some convenience functions for converting into and working on-top of this format in the /utils/ folder of the edfread distribution.

4.1. Fixmat Format

The fixmat format is a flat representation of fixation data, in contrast to the more hierarchical organisation of the edfread output. It is the prefered form of storage when working on your experiment data. The general aim is to make access to all fixations of an experiment fast and smooth. For this to reach the fixmat structure contains information about one fixation per column. The simplest possible variant looks like this:

```
fixmat = 10000x1 struct array with fields:
       start: [  0  200 .. 500]
         end: [100  300 .. 600]
           x: [320   40 .. 300]
           y: [240  220 .. 290]
         eye: [  1    1 ..   2]
   condition: [  0    0 ..  56]
SUBJECTINDEX: [  1    1 ..  12]
```

Where condition and SUBJECTINDEX are meta data fields as written by pytrack, start/end/x/y are fixation start, end time, x- and y-position, eye is 1 for the left and 2 for the right eye.

4.2. fixations.m

fixations.m takes a eyetracking+metadata pair as returned by edfread and returns a structure in fixmat format. It will automatically add all your meta data fields for the experiment and the single trials. These fields will be converted to the int32 type. If you have e.g. string or floating point data, you will want to modify fixations.m's initialization in init_struct() and add a custom conversion line in the main loop.

The eye is selected by comparing calibration results and checking for availbility of data. Fixations prior to stimulus onset will be discarded.

4.3. split.m

split.m helps splitting up a fixmat depending on it's field values. You can supply an arbitray number of field/value combinations to filter for:

```
split(fixmat, 'field1', values1, 'field2', values2, .. )
```

In the simple case, where length(values)==1 for all values, split.m will return a structure of the same format as fixmat, but only with those elements matching the field criteria. For example, to get all fixations of subject 20 in condition 1 from a fixmat of the whole experiment:

```
split(fixmat, 'condition', [1], 'SUBJECTINDEX', [20])
ans =
       start: [1x2360 int32]
         end: [1x2360 int32]
           x: [1x2360 single]
           y: [1x2360 single]
         eye: [1x2360 uint8]
   condition: [1x2360 int32]
SUBJECTINDEX: [1x2360 int32]
```

In contrast, when there is more than one value per field, you will get a seperate structure for each possible value:

```
>> result = split(fixmat, 'eye', [1, 2]);
>> result(1)
           start: [1x3095 int32]
             end: [1x3095 int32]
               x: [1x3095 single]
               y: [1x3095 single]
             eye: [1x3095 uint8]
    EXPERIMENTOR: [1x3095 int32]
    SUBJECTINDEX: [1x3095 int32]
       condition: [1x3095 int32]
            file: [1x3095 int32]

>> result(2)
           start: [1x1167 int32]
             end: [1x1167 int32]
               x: [1x1167 single]
               y: [1x1167 single]
             eye: [1x1167 uint8]
    EXPERIMENTOR: [1x1167 int32]
    SUBJECTINDEX: [1x1167 int32]
       condition: [1x1167 int32]
            file: [1x1167 int32]
```

4.4. concat.m

concat.m concatenates struct arrays together by field. You can use it to combine the results of successive invocations of edfread.

4.5. Sample Session

```
% sample usage of utility functions

% read data w/ edfread
[dat, meta] = edfread('/Users/jsteger/lagTest/AB_VP016.EDF')

% lets do some pre-processing:
% convert the experimentor's name to a numerical id
experimentors = {'johannes'; 'andreas'; 'dummy'}
[discard, discard, index] = intersect(meta.EXPERIMENTOR, experimentors);
% (index is now the index of the string 'meta.EXPERIMENTOR' in the cell
% array 'experimentors', e.g. 2 for 'andreas')
% note: fixations.m expects the string representation of an integer here,
% so we convert the index
meta.EXPERIMENTORS = num2str(index);

% now we convert to fixmat
fixmat0 = fixations(dat, meta);

% say we did this for two other subjects too and saved the results to
% fixmat1 and fixmat2 (in practice, you will of course want to do that
% within a function)
% we can now merge all data into one structure:
fixmat = concat(fixmat0, fixmat1, fixmat2);

% you should really save that fixmat now
save('precious.mat', 'fixmat');

% some use cases
% plot all fixations condition 0 and 1 with different colors
condfix = split(fixmat, 'condition', [0, 1]);

plot(condfix(1).x, condfix(1).y, '+r');
hold;
plot(condfix(2).x, condfix(2).y, '+b');
```

Chapter 5. Developer documentation

Note: This part of the manual is only relevant for you if you wish to modify the *source code* of edfread.

5.1. Introduction

Edfread is written in as a Matlab extension module in ANSI C/C++.

5.2. Tool-chain

Builds on all platforms are driven by a Makefile. The Linux version also builds the documentation (User-guide in html and pdf, overview in pdf, Matlab help in .m) and invokes remote shells for compilation on other platforms.

> **Requirements for compiling:** For Linux and Mac OS X, a recent gnu g++ compiler is sufficient. Windows requires Installation of the mingw32 environment with the gnu compiler collection.

Documentation is created with Doxygen (Developer Reference), Docbook (this manual) and Inkscape (Cheatsheet).

5.3. Libraries

Edfread uses the GNU Standard C Library for base datatypes, libedfapi by SR-Research for .edf file access and Matlab's C bindings.

5.4. Structure

Please see the separate "Edfread Reference Manual" (`dev-apiref.pdf`) for a detailed listing of available structures and functions.