Workshop on analysis of eye movement data in Matlab

Matteo Lisi

2025-03-05

Contents

Image classification task **2**
- Getting started . . . 3
- Import gaze-recording file . . . 4
- Data preparation . . . 9
- Visualizations (1) . . . 14
- Saccade & fixation analysis . . . 15
- Visualization (2) . . . 18

Image classification task

This task was a free-viewing task, in which participants observed some images, each presented for 3 seconds, and then made a decision about whether they were real photo or AI-generated. In this workshopw we aim to characterize the scan path by identifying saccades and fixations.

You can find the analysis script in the folder `analysis` in the Gihub repository (https://github.com/mattelisi/NeuroMethods/tree/master/eyetracking_tasks/image-classification/analysis).

We aim to obtain detailed information about the saccades (direction amplitude, speed, etc.) as well as the fixations (position and duration). One of the outcomes is creating scan-paths visualizations like the one below, where I ahve superimposed the gaze path (blue lines) onto the image, as well as added circle to mark individual fixations; note that the size of the circle is proportional to the fixation duration (larger circles indicate participant dwell on that for longer).

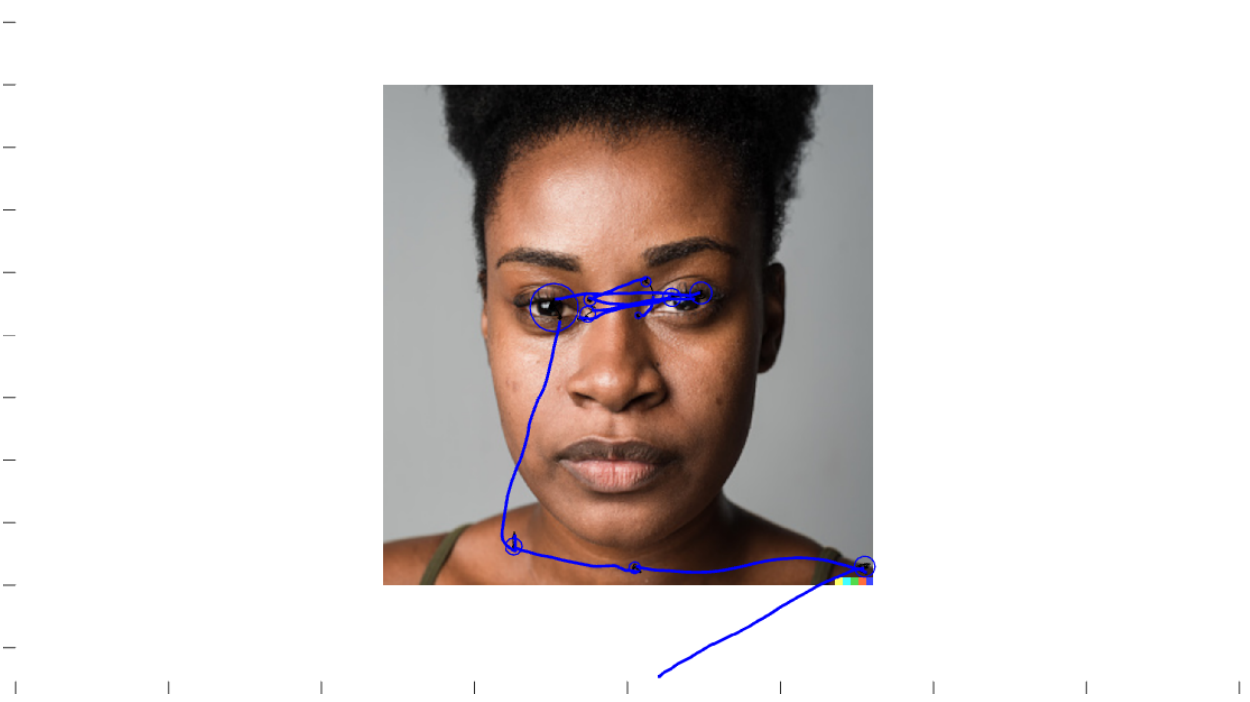


Figure 1: Scanpath of an observer looking at a face. We can see that most fixations tend to be around the eyes, known to be an informative part of the face for inferring other people's emotional and cognitive states

Getting started

We begin by clearing the workspace and loading some useful custom functions (these are contained in the folders `functions` and `analysis_functions`). We add these to Matlab path using the function `addpath` so that we can call the functions included in these folders as if they were built-in functions.

```
clear all

addpath('../functions');
addpath('./analysis_functions');
```

For our analysis we will need to use some information about the experimental setup - It's good practice to save them somewhere along with the eye movement recordings. We need in particular the screen size (width), the viewing distance and the screen resolution. These allow converting measures of length/size of eye movements and stimuli on screen from pixels to degrees of visual angle.

In particular the value `ppd` indicate how many pixels on the screen correspond to 1 degrees of visual angle.

```
scr.subDist = 80;   % subject distance (cm)
scr.width   = 570;  % monitor width (mm)
img_duration = 3;

scr.xres = 1920;
scr.yres = 1080;
scr.xCenter = scr.xres/2;
scr.yCenter = scr.yres/2 ;
ppd = va2pix(1,scr); % pixel per degree
```

It is computed from the function `va2pix` - if you open this (type `open va2pix` in the command window), you can see it is computed as

$$\text{ppd} = \text{distance} \times \tan\left(\frac{\pi}{180}\right) \frac{\text{xres}}{\text{width}}$$

The formula follows from trigonometry (see slides of math lectures), and indicates how many pixels corresponds to a rotation of the eye by one degree (starting from a central fixation and moving the gaze outward; here distance is the distance of the observer from the screen).

Practice tasks:

- ☐ Open the `va2pix` function and verify that it correspond to the above equation.

Import gaze-recording file

Next we import the data file. Here `.edf` stands for 'eyelink data file'. You can change the file name `../data/S1.edf` to import data from a different participant.

```
raw_data = '../data/S2301.edf';

% load eye movement file
ds = edfmex(raw_data);
```

Note that to import the edf file directly in Matlab you need to have installed the Eyelink API (you can find this link after registering to SR research forum).

If you don't have the Eyelink API installed, you can load directly the converted file (after downloading these from the Github repository) as follow:

```
load('S2301_edfstruct.mat')
```

Either way, you should have as a result a matlab structure loaded in your workspace:

Practice tasks:

- ☐ Type `ds` in the command window and examine what type of information you have loaded in memory. What kind of datatype is it? What do you think the difference fields contain?

FSAMPLE

The actual gaze recordings, that is the X (horizontal) and y (vertical) coordinates of gaze on screen are in the **FSAMPLE** field, among other things.

Practice tasks:

- ☐ Type ds.FSAMPLE‘ in the command window. What do you see? What type of information do you think we need to retrieve for the analysis?

This table, from the documentation of `edfmex`, provide information about what data is contained in each field (taken from https://www.sr-research.com/support/thread-28.html and also provided in the Github folder at this https://github.com/mattelisi/NeuroMethods/blob/master/workshops/eye_movement_analysis/EDF_Data_Structure.pdf)

FSAMPLE:

The **FSAMPLE** structure holds information for each sample in the EDF file. Depending on the recording options, some of the fields may be empty.

Field	ROW 1	ROW 2	Notes
time	time stamp of sample (in milliseconds)		
px	left eye raw pupil x coordinates	right eye raw pupil x coordinates	Raw pupil-center position (or pupil minus CR if running in pupil-CR mode)
py	left eye raw pupil y coordinates	right eye raw pupil y coordinates	See above
hx	left eye head-referenced eye position x	right eye head-referenced eye position x	HREF data is related to the tangent of the rotation angle of the eye relative to the head.
hy	left eye head-referenced eye position y	right eye head-referenced eye position y	See above
pa	left eye pupil size data	right eye pupil size data	This value is uncalibrated and based on diameter or area (depending on which was chosen during data collection)
gx	X Gaze position for Left Eye	X Gaze position for Right Eye	Gaze in screen pixel co-ordinates
gy	Y Gaze position for Left Eye	Y Gaze position for Right Eye	See above
rx	left eye PPDx	right eye PPDx	Pixels per degree
ry	left eye PPDy	right eye PPDy	Pixels per degree
gxvel	left eye gaze velocity x	right eye gaze velocity x	Value in degrees of visual angle per second
gyvel	left eye gaze velocity y	right eye gaze velocity y	Value in degrees of visual angle per second
hxvel	left eye head-referenced eye velocity x	right eye head-referenced eye velocity x	See hx above
hyvel	left eye head-referenced eye velocity y	right eye head-referenced eye velocity y	See hy above
rxvel	left eye raw x velocity	righ eye raw x velocity	In degs per second
ryvel	left eye raw y velocity	right eye raw y velocity	In degs per second
fgxvel	left eye fast gaze x velocity	right eye fast gaze x velocity	fast = uses fast velocity model
fgyvel	left eye fast gaze y velocity	right eye fast gaze y velocity	See above
fhxvel	left eye fast head-referenced velocity x	right eye fast head-referenced velocity x	See above
fhyvel	left eye fast head-referenced velocity y	right eye fast head-referenced velocity y	See above
frxvel	left eye fast head-referenced ppdx	right eye fast head-referenced ppdx	See above
fryvel	left eye fast head-referenced ppdy	right eye fast head-referenced ppdy	See above
hdata	head-tracker data		
flags	flags to indicate contents		
input	extra (input word)		Unused
buttons	button state and changes		From button box attached to Host PC
htype	head tracker data type		
errors	process error flags		

The most important of these are the `time`, which provides the time at which each sample of gaze position was recorded, and `gx` which are the coordinates of gaze on the screen (these are in screen coordinates, i.e. pixels measured from the top-left corner of the screen).

`FEVENT`

Similarly the, `FEVENT` field contains information about 'events' - these are timestamped *events* of the experiment, such as the onset time of the visual stimulus, or the response time. These are necessary because for most analysis we want to align in time the gaze recording with task events. The fields here are the following (see the documentation for full description of these)

Practice tasks:

- ☐ Type ds.FEVENT‘ and examine the field names.

Out of these fields, in particular, the `message` field contains events that we set our code to record. For exmaple, if you look at the code for this experiments, you will find at some point the line:

```
Eyelink('message', 'TRIAL_START %d', t);
```

This command was executed during the experiment at the beginning of each trial to indicate that trial number `t` was starting. The Eyelink system can record with msec precision the time at which the command was executed, and make a note ("timestamp") in the gaze recordings.

If we look at the messages in the data file, we can see for example that for this specific data file the 70-th message[1] correspond to the start of the first trial

```
>> ds.FEVENT(70).message

ans =

    'TRIAL_START 1'
```

We can find out the precise time of when that trial started (the "timestamp") by looking at the `sttime` field of the 70-th event.

```
>> ds.FEVENT(70).sttime

ans =

  uint32

   404996
```

Practice tasks:

- ☐ Try seeing what other types of messages are included, by printing a range of messages instead of single one. One simple way to do it is by using e.g. `ds.FEVENT(100:200).message` for retrieving messages all the messages between the 100-th and the 200-th.
- ☐ Try and retrieve the time of some other event.

The number we get is the timestamp (time in milliseconds since the computer was switched on)[2].

The events allows us to get useful information, for example how many trials did this participant complete. Top do so, I just count how many times a certain message occur (for example here I do this with the message `EVENT_FixationDot` which in the code of the experiment is what signalled the appearance of the image on screen).

Here I use the function `find` and `strcmp` to get the indexes of all messages indicating that an image appeared on screen:

```
>> find(strcmp({ds.FEVENT.message}, 'EVENT_FixationDot')==1)

ans =
```

[1] It may be a different message if you have a different file, or perhaps an empty string

[2] The `unit32` indicator simply tells us that this number is encoded by matlab as a 32-bit unsigned integer. This allows matlab to save some memory as a `unit32` occupies only 4 bytes of memory, as opposed to the 8 bytes of a `double` type. See https://uk.mathworks.com/help/matlab/matlab_prog/strategies-for-efficient-use-of-memory.html for more information on memory usage and data types in Matlab.

```
    72   145   222   309   370
```

We have 5 numbers, indicating that this participant completed 5 trials.

Data preparation

Here we put the data in a format more suitable for further data analysis.

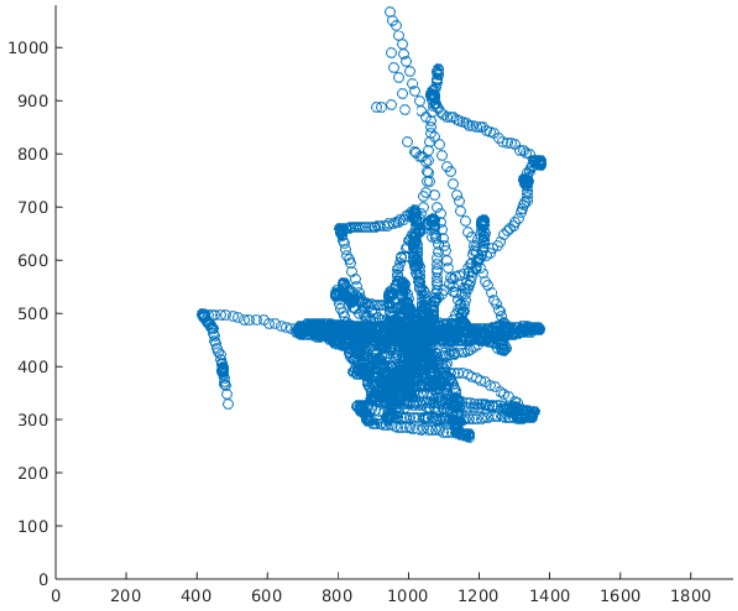
First, we set information about which eye was tracked:

```
% which eye was tracked?
% 0=left 1=right (add 1 for indexing below)
eye_tracked = 1 + mode([ds.FEVENT.eye]);
```

This is required because the `gx` field contain a 2-column vector, although only one column contain useful data. The reason for this is that the eyelink could also be used to track binocular eye movements, using a slightly more involved setup. In this case we recorded data monocularly, and therefore one column is empty.

In case you are in doubt, you can plot the raw data and see what comes out. Here the data is in the second column, indicating that the right eye was tracked.

```
scatter(ds.FSAMPLE.gx(2,:),ds.FSAMPLE.gy(2,:))
xlim([0 scr.xres])
ylim([0 scr.yres])
```



Note that I have restricted the range of the plot to the screen coordinates: this is because when there are missing data point, (e.g. blinks) the edfmex program that imported the data automatically enter a very large value (100000000), and plotting these large values would make the figure unreadable.

Practice tasks:

- ☐ Try reproducing the plot above without the axis limits.
- ☐ Try to load and plot data from another participant.

Next, we need to write some code that loop over all trials and retrieve all information useful for the analysis, and put it in a format that is easier to work with. There are no hard and fast rules about how to do this, I personally find it useful to split the data according to trials, where for each trial I can easily find the gaze recording and other trials information (in this case, for example, the image presented, whether it was an AI-generated one, the response of the participant, and so on).

More specifically, in this case we will end up with the following structure:

```
>> ds2.trial

ans =

  1×5 struct array with fields:

    trial_n
    t_start
    t_end
    img_onset
    img_name
    fake_image
    accuracy
    imgRect
    timestamp
    eye_x
    eye_y
```

In order to create this structure, we will perform the following operations:

1. loop through all recorded events (i.e. the elements of `ds.FEVENT`);
2. identify the start of each trial and timing of key events for that trial; that is the `TRIAL_START` message, and others like the `EVENT_FixationDot` which indicates the time at which the image appeared on screen^[Note that there I didn't put message for the offset of the image, but we know it was 3 seconds later);
3. extract x and y coordinates of gaze (from `FSAMPLE.gx` and `FSAMPLE.gy`) for the current trial, by selecting gaze positions recorded after the start and before the end of the current trial;
4. remove missing values (e.g. replace the 100000000 introduced for missing values with `NaN` so as we don't risk erroneously consider them "true" recorded gaze positions).
5. store any other relevant information for the analysis of the current trial - such as the screen coordinates at which the image was presented, so that we can then spatially align gaze positions with the image.

Here is the code that run these operations:

```
% initialize & prea-allocate empty values for all trial information
trial_n = NaN;
trial_n_2 = NaN;
trial_n_3 = NaN;
t_start = NaN;
t_end =  NaN;
img_onset =  NaN;
img_offset =  NaN;
img_name =  NaN;
fake_image =  NaN;
accuracy =  NaN;
imgRect =  [];
timestamp  =  [];
```

```
eye_x =  [];
eye_y =  [];

ds2 = {};
trial_count = 0;
for i = 1:length(ds.FEVENT) % loop over all events

    % go through the list of events and extract relevant informations
    if ~isempty(ds.FEVENT(i).message)

        % parse string in different "words"
        sa = strread(ds.FEVENT(i).message,'%s');

        % message indicating the onset of trial
        if strcmp(sa(1),'TRIAL_START')
            trial_n = str2double(sa(2));
            t_start = ds.FEVENT(i).sttime;
        end

        % message indicating when the image appeared
        if strcmp(sa(1),'EVENT_FixationDot')
           img_onset = int32(ds.FEVENT(i).sttime);
           img_offset = img_onset + 3000; % image disappeared 3sec later
        end

        % message indicating the end of eye movement recoding
        if strcmp(sa(1),'TRIAL_END')
           trial_n_2 = str2double(sa(2));
           t_end = ds.FEVENT(i).sttime;
        end

        % extract all data info from the 'TrialData' message
        if strcmp(sa(1),'TrialData')

           img_name = sa(5);
           trial_n_3 = str2double(sa(2));
           fake_image = str2double(sa(4));
           accuracy = str2double(sa(7));
           imgRect = [str2double(sa(8:11))]';
        end
    end

    % if we have everything, then extract gaze position samples
    if ~isnan(trial_n) && ~isnan(trial_n_2) && ~isnan(trial_n_3)

        trial_count = trial_count+1;

        % find onset-offset of relevant recording
        index_start = find(ds.FSAMPLE.time==t_start);
        index_end = find(ds.FSAMPLE.time==img_offset + 400);

        % timestamp - note that we subtract img_onset such that time counting
        % will start when the image appeaared on the screen
        timestamp  =  int32(ds.FSAMPLE.time(index_start:index_end)) - img_onset;
```

```
        eye_x =  ds.FSAMPLE.gx(eye_tracked, index_start:index_end);
        eye_y =  ds.FSAMPLE.gy(eye_tracked, index_start:index_end);

        % remove missing values
        eye_x(eye_x==100000000 | eye_y==100000000) = NaN;
        eye_y(eye_y==100000000 | eye_x==100000000) = NaN;

        % remove gaze coordinates outside of screen (these could be blinks)
        eye_x(eye_x<0 | eye_x>scr.xres | eye_y<0 | eye_y>scr.yres ) = NaN;
        eye_y(eye_y<0 | eye_y>scr.yres | eye_x<0 | eye_x>scr.xres ) = NaN;

        % save everything into a single structure
        ds2.trial(trial_count).trial_n = trial_n;
        ds2.trial(trial_count).t_start = t_start;
        ds2.trial(trial_count).t_end = t_end;
        ds2.trial(trial_count).img_onset = img_onset;
        ds2.trial(trial_count).img_name = img_name;
        ds2.trial(trial_count).fake_image = fake_image;
        ds2.trial(trial_count).accuracy = accuracy;
        ds2.trial(trial_count).imgRect =  imgRect;
        ds2.trial(trial_count).timestamp  =  timestamp  ;
        ds2.trial(trial_count).eye_x =  eye_x;
        ds2.trial(trial_count).eye_y =  eye_y;

        % re-initialize
        trial_n = NaN;
        trial_n_2 = NaN;
        trial_n_3 = NaN;
        t_start = NaN;
        t_end =  NaN;
        img_onset =  NaN;
        img_offset =  NaN;
        img_name =  NaN;
        fake_image =  NaN;
        accuracy =  NaN;
        imgRect =  [];
        timestamp  =  [];
        eye_x =  [];
        eye_y =  [];

    end

end
```

Looking at the code above, you may have noticed that I retrieve the trial number 3 times, from 3 distinct messages (`TRIAL_START`, `TRIAL_END`, and `TrialData`). This type of redundancies are not strictly necessary, but are often helpful to catch programming errors. In this case it allows me to be sure I have everything before proceeding with the analysis of a trial.

`TrialData`

Another thing to notice is that I use the message `TrialData` to store a lot of information about the trial (this is a message that was included in the eye movement recording at the end of each trial).

Here is an example:

```
>> ds.FEVENT(459).message

ans =

    'TrialData 5    S1  0   woman_real.png  82  1   584 164 1.336589e+03    9.165889e+02
```

If you check the code of the experiment, this message was sent using the following commands

```
% collect trial & response information
data = sprintf('%i\t%s\t%i\t%i\t%i\t%i\t%i\t%i\t', is_fake, char(img_i), response, correct, imgRect);
dataStr = sprintf('%i\t%s%s\n',t,info_str,data); % print data to string

% write data to edfFile
Eyelink('message', 'TrialData %s', dataStr);
```

Practice tasks:

- ☐ Try and print in the command window the `TrialData` from a different trial.

Visualizations (1)

Having reformatted the data as described in the previous section is useful, for example it makes it much easier to create visualizations.

Here are some things we can do. First, select a trial

```
% select a trial number - here I just take the first one
t = 1;
```

You can visualize the image presented in that trial using:

```
% retrieve path to image file
if ds2.trial(t).fake_image ==1 % was it AI generated?
    imgpath = ['../img/fake/', char(ds2.trial(t).img_name)];
else
    imgpath = ['../img/real/', char(ds2.trial(t).img_name)];
end

% open a figure and add the image in a subpanel
figure;
subplot(1,3,1);
imshow(imgpath);
```

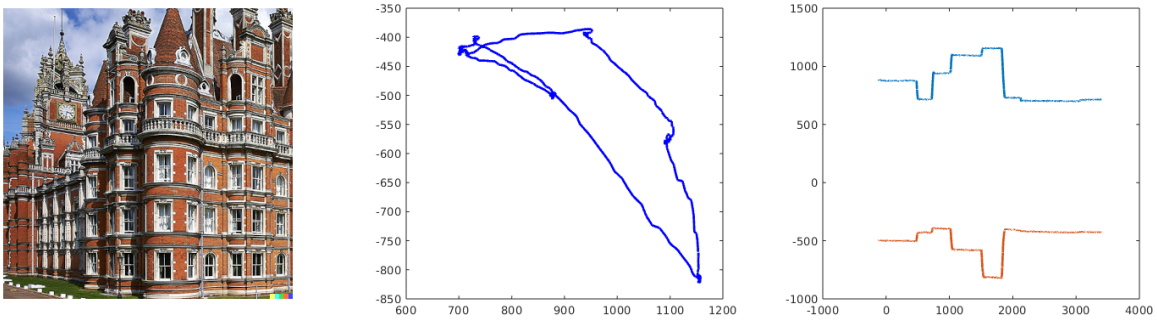
We can add a plot of gaze positions (x and y coordinates expressed in pixels) in another subpanel

```
subplot(1,3,2);
XY = [ds2.trial(t).eye_x; -ds2.trial(t).eye_y]';
plot(XY(:,1), XY(:,2), 'b', 'LineWidth', 2);
```

Now add a plot of position (both x and y) as a function of time

```
subplot(1,3,2);
XY = [ds2.trial(t).eye_x; -ds2.trial(t).eye_y]';
plot(XY(:,1), XY(:,2), 'b', 'LineWidth', 2);
```

Result:



Practice tasks:

- ☐ Make a similar plot but for at least another trial.
- ☐ Try and think about ways in which you could improve the informativeness of the plot.

Saccade & fixation analysis

In order to identify saccades onsets and offsets, we use an algorithm developed by Ralf Engbert and Konstantin Mergenthaler[3]. We won't be examining the algorithm in detail, for our purposes it is sufficient to say that it detects onsets and offsets based on gaze velocity (e.g. saccade onsets are detected when the speed of gaze movement exceed a certain threshold), but has some advantages compared to other algorithms (included the ones built-in in commercial software) that makes it more accurate and able to detect also small micro-saccades with amplitudes less than 1 degree of visual angle.

This algorithm requires setting some parameters that are defined in the following bit of the code:

```
% select trial to be analysed
t = 5;

% saccade algorithm parameters
SAMPRATE  = 1000;       % Eyetracker sampling rate
velSD     = 5;          % lambda for microsaccade detectionc
minDur    = 8;          % threshold duration for microsaccades (ms)
VELTYPE   = 2;          % velocity type for saccade detection
maxMSAmp  = 1;          % maximum microsaccade amplitude
mergeInt  = 10;         % merge interval for subsequent saccadic events
```

Now extract the gaze positions for the desired trials, filter them and differentiate them to obtain velocity.

```
xrs = [];
xrsf = [];
vrs = [];
vrsf= [];

% gaze position samples
XY = [ds2.trial(t).eye_x; ds2.trial(t).eye_y]';

% invert Y coordinates and transform in degrees relative to screen center
xrsf = double((1/ppd) * [XY(:,1)-scr.xCenter, (scr.yCenter-XY(:,2))]);

% filter gaze position data (mostly for plotting)
xrs(:,1) = movmean(xrsf(:,1),6);
xrs(:,2) = movmean(xrsf(:,2),6);

% compute velocities
vrs = vecvel(xrs, SAMPRATE, VELTYPE);    % velocities
vrsf= vecvel(xrsf, SAMPRATE, VELTYPE);   % velocities
```

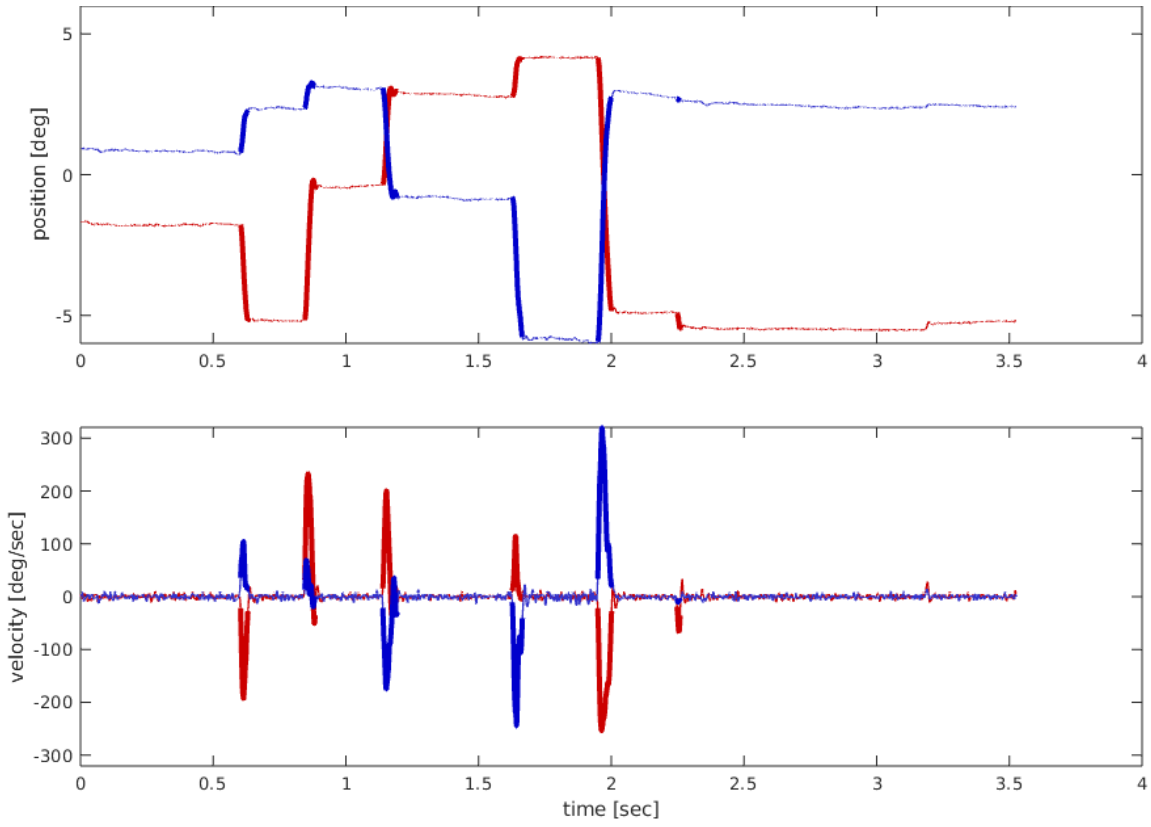
Finally, use the algorithm to compute saccade onsets and offsets.

```
% compute saccade parameters
mrs = microsaccMerge(xrsf,vrsf,velSD, minDur, mergeInt);  % saccades
mrs = saccpar(mrs);
```

Now that we have parsed the saccades, we can annotate plots, for example by highlighting the part of gaze recordings correspondign to saccadic eye movements (shown with a thicker line in the next figure)

[3]Engbert, R., & Mergenthaler, K. (2006). Microsaccades are triggered by low retinal image slip. *Proceedings of the National Academy of Sciences, 103*(18), 7192–7197. https://doi.org/10.1073/pnas.0509557103

```
% PLOT TRACES
% prepare figure and axes
close all;
cbac = [1.0 1.0 1.0];
h1 = figure;
set(gcf,'color',cbac);
ax(1) = axes('pos',[0.1 0.6 0.85 0.4]); % left bottom width height
ax(2) = axes('pos',[0.1 0.1 0.85 0.4]);

% format timers
timers = double(ds2.trial(t).timestamp);
timeIndex = (timers - timers(1) +1)/1000;

axes(ax(1));
% plot horizontal position
plot(timeIndex,xrs(:,1),'-','color',[0.8 0 0],'linewidth',1);
hold on
for i = 1:size(mrs,1)
    plot(timeIndex((mrs(i,1):mrs(i,2))), xrs(mrs(i,1):mrs(i,2),1),'-','color',[0.8 0 0],'linewidth',4);
end
% plot vertical position
plot(timeIndex,xrs(:,2),'-','color',[0.2 0.2 0.8],'linewidth',1);
for i = 1:size(mrs,1)
    plot(timeIndex((mrs(i,1):mrs(i,2))), xrs(mrs(i,1):mrs(i,2),2),'-','color',[0 0 0.8],'linewidth',4);
end
ylim([-max(abs(xrs(:))),max(abs(xrs(:)))])
ylabel('position [deg]');

axes(ax(2));
% plot horizontal position
plot(timeIndex,vrs(:,1),'-','color',[0.8 0 0],'linewidth',1);
hold on
for i = 1:size(mrs,1)
    plot(timeIndex((mrs(i,1):mrs(i,2))), vrs(mrs(i,1):mrs(i,2),1),'-','color',[0.8 0 0],'linewidth',4);
end
% plot vertical position
plot(timeIndex,vrs(:,2),'-','color',[0.2 0.2 0.8],'linewidth',1);
for i = 1:size(mrs,1)
    plot(timeIndex((mrs(i,1):mrs(i,2))), vrs(mrs(i,1):mrs(i,2),2),'-','color',[0 0 0.8],'linewidth',4);
end
ylim([-max(abs(vrs(:))),max(abs(vrs(:)))])

xlabel('time [sec]');
ylabel('velocity [deg/sec]');
```



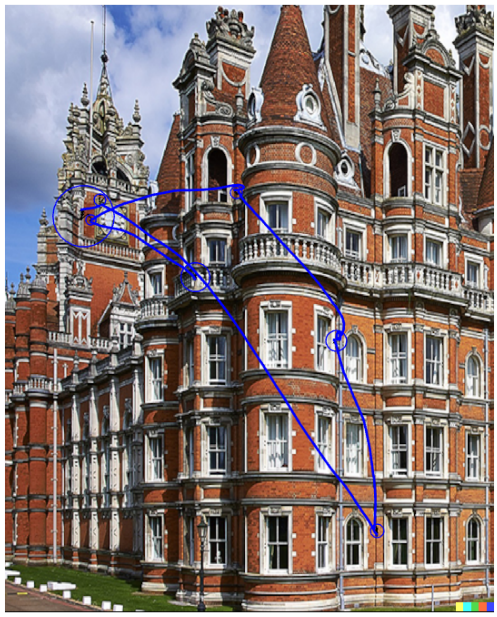
Practice tasks:

- ☐ Try reproducing the plot above using data from another trial.
- ☐ Examine the object `mrs` that has been created as part of this analysis. What does it contain? Hint: check the comments in the files of the function `microsaccMerge` and `saccpar` (e.g. by typing: `open microsaccMerge`). What do these two functions do?

Visualization (2)

```
h2 = figure;
figure(h2);
cbac = [1.0 1.0 1.0, 0];
set(gcf,'color',cbac);

if ds2.trial(t).fake_image ==1
    imgpath = ['../img/fake/', char(ds2.trial(t).img_name)];
else
    imgpath = ['../img/real/', char(ds2.trial(t).img_name)];
end
C = imread(imgpath);

img_c = ds2.trial(t).imgRect ./ [scr.xres,scr.yres,scr.xres,scr.yres];
img_c = [img_c(1), img_c(2), img_c(3)-img_c(1), img_c(4)-img_c(2)];
ax_img = axes('pos',img_c);

%C = C(end : -1: 1, :, :);
image(ax_img , C)
axis off

ax_gaze = axes('pos',[0 0 1 1],'Color','none'); % left bottom width height
axes(ax_gaze);
hold on

display_x = [-(scr.xres/2)/ppd, (scr.xres/2)/ppd];
display_y = [-(scr.yres/2)/ppd, (scr.yres/2)/ppd];

plot(ax_gaze , xrs(:,1),xrs(:,2),'k-');
xlim(display_x);
ylim(display_y);

for i = 1:size(mrs,1)
    plot(xrs(mrs(i,1):mrs(i,2),1),xrs(mrs(i,1):mrs(i,2),2),'b-','linewidth',2);
end
if ~isempty(mrs)
    plot([xrs(mrs(:,1),1) xrs(mrs(:,2),1)]',[xrs(mrs(:,1),2) xrs(mrs(:,2),2)]','b.');
end

% make a table of fixations
n_fix = size(mrs,1) + 1;
fixtab = zeros(n_fix, 5);
for i = 1:n_fix

    fix_start = NaN;
    fix_end = NaN;
    fix_dur = NaN;
    fix_coord = NaN;

    if i==1
        fix_start = 1; %ds2.trial(t).timestamp(1);
    else
```

```
        fix_start = mrs(i-1,2)+1;
    end

    if i==n_fix
        fix_end = length(ds2.trial(t).timestamp); %(end);
    else
        fix_end = mrs(i,1)-1;
    end

    fix_dur = fix_end - fix_start;

    fix_coord = [nanmean(xrs(fix_start:fix_end,1)), nanmean(xrs(fix_start:fix_end,2))];

    fixtab(i,:) = [fix_start,fix_end,fix_dur,fix_coord];
end

for i = 1:n_fix
    plot(fixtab(i,4), fixtab(i,5),'o','color',[0 0 1],'markersize',fixtab(i,3)/25,'linewidth',1);
end
```



Practice tasks:

- ☐ Examine the code to produce the last plot, step by step. Think whether you understand the logic of the code.

☐ Reproduce and save this plot for at least another trial.