

Pupillometry practical:

Using the pupil to track memory processes

This practical, which will count towards your workshop assignments grade, will showcase how you may collect *and* analyze pupillometric data to learn about working memory.

The pupil responds to brightness: it constricts (i.e., becomes smaller) when light enters the eye, whereas it dilates (i.e., becomes bigger) in darkness. Classically it has been regarded as a low-level reflexive mechanism that doesn't involve higher-order cognition. In the past few decades, this view has changed. For example, researchers have recently shown that this so-called pupillary light response does not only apply to the things at which we look directly, but also to the things to which we attend covertly (i.e., without looking) (e.g., Mathôt, van der Linden, Grainger, & Vitu, 2013). It has also been shown that the pupil responds to the brightness of *internal* representations held in (working) memory. This practical revolves around these phenomena.

We will build a pupillometric experiment in which participants have to memorize, in each trial, the orientation of two gabor stimuli (Figure 1). The two stimuli are presented on respectively a white and a black background. After a short duration, the participant sees a third gabor stimulus, and the participant has to indicate whether this final stimulus matches with one of the two stimuli seen earlier.

If the pupil indeed responds to the brightness of memorized objects, then we may expect to find that the pupil's size is influenced by whether the probed stimulus was perceived on a white or black background.

This practical consists of two parts. First, we'll build the experiment. In the second part, we'll analyze simulated pupillometric data.

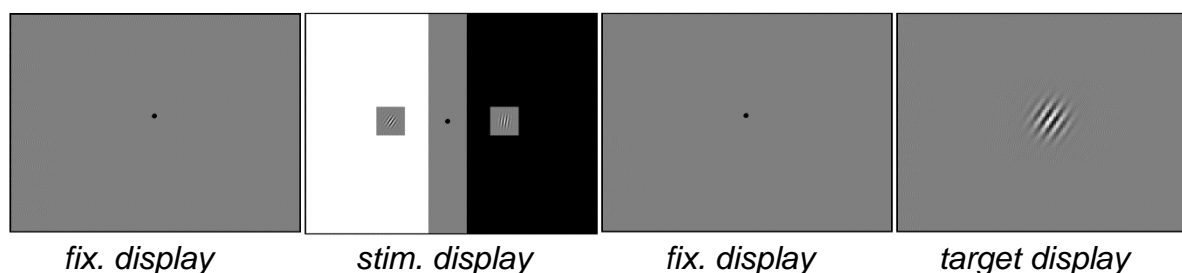





Figure 1. Trial sequence. The participant fixates on the screen center. Two gabor patches are shown left and right of fixation. The white and black coloring of the background is varied across trials.



Part I: Building the Experiment


First of all, make sure that the PyGaze package is installed with your version of OpenSesame. It should be installed with the latest Windows versions. If not, see for instructions: <https://osdoc.cogsci.nl/3.2/manual/eyetracking/pygaze/#install-from-source>.

1.1 Laying the fundamentals

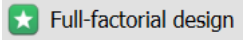
Open a default template in OpenSesame, and delete the **getting_started** and **welcome** items. As we'll do eye-tracking, place a **PyGaze_init**  item at the top of the experiment. If you do not see this item, then press the downward pointing arrow at the bottom-left corner of the screen to expand the item window. Clicking on **pygaze_init**, set "Select tracker type" to *Simple dummy*. This allows us to run the experiment without having an actual eye-tracker connected to our computer.

As with our previous experiments, we'll use a loop item to implement our experimental conditions. Place a **loop**  item below **pygaze_init**. In the **loop** item, place a **sequence**  item and name it *trial_sequence*.

Eye-tracking (and pupillometry) in OpenSesame requires the use of but a few items. Here, our trial sequence will have to comprise **pygaze_start_recording**  and **pygaze_stop_recording** .

At the bottom of the sequence item (i.e., after having stopped recording), you can place a **logger**  item; we want to collect data, after all.


1.2 Specifying conditions




As indicated earlier, participants will have to indicate whether or not a target stimulus had been seen before. Targets will have been seen either to the left or right of fixation. Furthermore, the target's background will have been either black or white. Our experiment thus follows a **2 x 2 x 2** factorial design. In the **loop** item, click on the  **Full-factorial design** button. Here, define the three columns *target_side* (with values *left* and *right* specified below it), *left_background* (with values *white* and *black* specified below it) and *final_stimulus* (with values *target* and *foil*) specified below it. Note that the value *foil* for *final_stimulus* will represent trials in which the stimulus was not one of the earlier stimuli (requiring a 'no' response from the participant). Press the 'OK' button, so that a total of 8 conditions are revealed in the **loop** item. Let's say that we want to test each of these conditions 10 times per participant: i.e., set *Repeat each cycle* to 10.

To make life easier later on, let's not only use a *left_background* column, but also a *right_background* column. Just enter the name *right_background* at the top of a fourth column, and enter values *white* and *black* below it, depending on whether the value in *left_background* is *black* or *white*, respectively (so the two are always different).

Finally, you may recall from previous practicals that we can indicate a `correct_response` here, which will allow the experiment to automatically classify responses as correct or incorrect. Thus, create a fifth column with the name `correct_response`. On each row, enter the value *up* or *down*, depending on whether the value for *final_stimulus* in that condition is *target* or *foil*, respectively.

1.3 Stimulus display

Now, let's start building our stimulus displays. As you have seen in Figure 1, potential target stimuli are shown on black or white backgrounds, while gray backgrounds are used for 'neutral' zones (e.g., the location of the participant's fixation). Therefore, click on the top item  in the overview area, and then change the *Background* parameter from *black* to *gray*. Similarly, change the *Foreground* parameter from *white* to *black*.

In `trial_sequence` , after `start_recording` , place a `sketchpad`  item and call it *stim_display*. When clicking on this item, you can see a bunch of drawing tools on the left. The fourth of these allows you to draw a fixation dot. Use it to place a fixation dot in the middle (again, take Figure 1 for reference). Now, we want the display to be part white and part black, divided by a band of gray in the middle, and we want to switch the color of these backgrounds from trial to trial.

Among the sketchpad tools, you'll find the *Draw rect element* tool (with square icon). Clicking on this tool, and then ticking the *fill* box above the drawing area, you can draw two large rectangles that fill almost half of the display (save for the gray band in the middle; see Figure 2). Right-clicking on each of these rectangles, select *edit* and then for the color parameter specify *color* = *[left_background]* and *color* = *[right_background]*, for the left and right rectangle, respectively. As such, the rectangles take on the colors that are specified in the `loop` item.

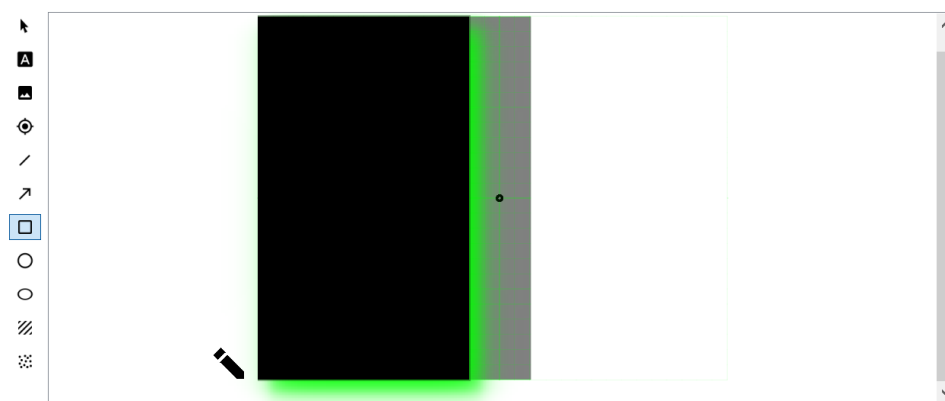


Figure 2. Drawing rectangles.

Now, we need to fetch ourselves some gabor patches. These are provided by the penultimate of the sketchpad tools. Just as with a fixation dot, you can click anywhere on the screen to place a gabor patch. Note that as your mouse hovers across the screen, at the top right corner you see the (x,y) coordinates of the cursor. Use these to place gabors

neatly at (-192,0) and (192,0). Upon clicking, you'll see a pop-up window asking you for a bunch of parameters. Just ignore these for now and click 'OK'.

You'll now see the two gabor patches on the left and right of the fixation dot. Right-click these, and click on *edit*. You'll see a bunch of parameters, among which *orient* and *show if*. For the leftside gabor, specify *orient* = *[target_orientation]* and *show if* = *[target_side] = left*. For the rightside gabor, specify *orient* = *[distractor_orientation]* and again *show if* = *[target_side] = left*. You may have realized that we've only created stimuli for the conditions in which the target stimulus is shown on the left (and an irrelevant 'filler' or 'distractor' stimulus is shown on the right). We also need a pair of gabors, at the same locations, for the conditions in which a target stimulus is shown on the right. Hence, place two new gabors at the same two locations, and choose the appropriate values for *orient* and *show if*.

Clearly, we haven't created the variables *target_orientation* and *distractor_orientation* (which, as you may guess, should indicate the orientation of the gabors) yet. This will be done soon enough. First, however, let's create our final stimulus display.


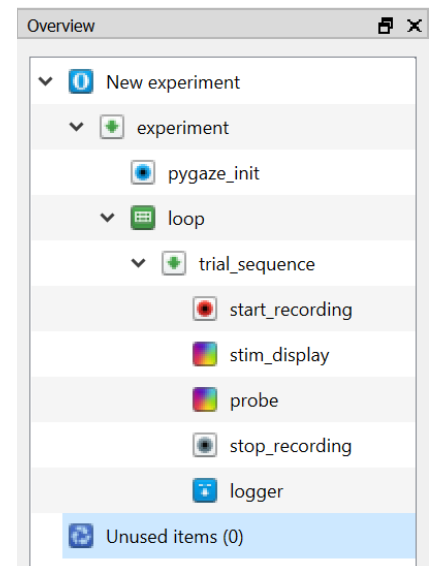

Drag another **sketchpad**  (called *probe*) into **trial_sequence** and place a gabor in its center. Again, right-click the gabor to edit it, and specify *orient* = *[target_orientation]*, and *show if* = *[final_stimulus] = target*. This stimulus will cover those trials in which the gabor was shown before. On the other half of trials, the participant should see a gabor that wasn't shown before. Therefore, place a second gabor at the same location, edit it, and this time enter the values *orient* = *[foil_orientation]* and *show if* = *[final_stimulus] = foil*. The experiment overview should now look as shown in Figure 3.

Figure 3. Experiment overview.



1.4 Specifying gabor orientations

We haven't specified the orientations of our gabors in the **loop** item. The main reason for this is that the gabor can have 360 different orientations (or, technically, 180, since a 90 degree gabor and a 270 degree gabor look the same). If we'd allow all these orientations in the loop item, we'd end up with way too many trials. Therefore, let's instead determine the orientations randomly.

Place an **inline_script**  item, named *choose_orientations*, at the top of the **trial_sequence**. In the **prepare phase**, enter the following lines of code:

```
import random
all_orientations = range(0,180)
exp.target_orientation = random.choice(all_orientations)
```

The first line makes certain Python functions (related to randomness) available. The second line creates a list (called 'all_orientations') with values between 0 and 180. With the third line, we create a variable ('exp.target_orientation'), the value of which is a randomly chosen value from all_orientations.

Note that we can't simply do the exact same thing for *distractor_orientation* and *foil_orientation*. Why? Well, because we don't want to run the risk that we accidentally pick the same values for these variables. That is, *target_orientation*, *distractor_orientation* and *foil_orientation* have to be sufficiently distinct.

Therefore, want to do the following. We want to pick a random value from 'all_orientations' for a variable exp.distractor_orientation, and then we want to check if that value is sufficiently distinct (say, be at least 20 degrees different) from exp.target_orientation. If this is not the case, we want to pick the value again (and again and again, until our requirement is met).

To do something like this repeatedly, we can initiate a loop:

```
while True:
    exp.distractor_orientation = random.choice(all_orientations)
    if exp.distractor_orientation not in range(exp.target_orientation-
20,exp.target_orientation+20):
        break
```


What happens here? 'While True' initiates an endless loop (so it doesn't end after, for instance, a fixed 10 cycles). In the loop, we pick a value for exp.distractor_orientation. We subsequently check whether the value is *not* within the range of 40 degrees around the target_orientation. If this requirement is met, we *break* from the loop (i.e., we exit it). If the condition is not met, we simply return to the start of the loop.

Now we need to repeat this trick for exp.foil_orientation, but this time we need to make sure that its value is sufficiently distinct from both exp.target_orientation and exp.distractor_orientation:

```
while True:
    exp.foil_orientation = random.choice(all_orientations)
    if exp.foil_orientation not in range(exp.target_orientation-
20,exp.target_orientation+20) and exp.foil_orientation not in
range(exp.distractor_orientation-20,exp.distractor_orientation+20):
        break
```


And that covers our gabor orientations.


1.5 Finishing up the trial_sequence

The trial sequence is still lacking a **keyboard response**  item. Place this right after the *probe sketchpad* item. In the *Allowed responses* field, enter *up;down*. You may recall that our *Correct response* is already specified in the *loop* item.

Now, we just need some empty fixation displays among the various items that we already put in place. For instance, before the onset of the two gabors, it would be good to have a fixation display (a sketchpad with only a fixation dot in the center). Right after *stim_display*, we want another fixation display.

We also need to take into consideration that pupil responses are rather slow and sluggish: they need at least approximately a second to unfold. Therefore, we don't want the participant to respond (and the trial to end) too quickly upon showing the *probe* display. Therefore, set the duration of *probe* to 1000 ms. Set the duration of *stim_display* to 1000 ms too. Set the duration of the other displays to 600 ms.

Since the duration of *probe* is now set to 1000 ms, the participant will not be able to respond until 1000ms after *probe's* onset. To let the participant know that he/she can respond, let's add an extra **sketchpad**  item in between *probe* and the **keyboard response**. Call it *response_display*, place a simple question mark in the center (using the text tool), and set the duration to 0ms. The latter is done so that we move on to the keyboard item immediately after the question mark appears.

Finally, place a last **sketchpad**  at the end of the trial sequence, and set its duration to 600 ms. This is just to create some time between trials, to allow the pupil's size to regress back to baseline. The experiment overview should now look as in Figure 4.

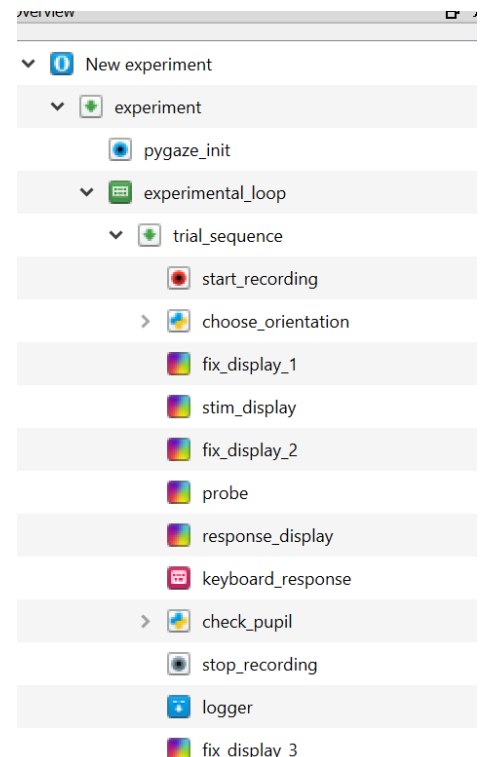




Figure 4.

1.6 Collecting pupil responses

In Figure 4, you see that there is yet another **inline_script**  item, called *check_pupil* in *trial_sequence*. You can guess what we're about to do with that script. However, just observing the pupil size at that time isn't the whole story. Why? Well, we don't know whether a pupil of, say, 16mm, is large or small. That is, we need a *baseline* measurement at the start of the trial. If at the start of the trial we observed a pupil of 12 mm, then we can say that the pupil dilated. If at the start the pupil was 19 mm, then we can say that it constricted.

So, in the **run phase** of the *choose_orientation* script, place the following line:

```
start_pupil = eyetracker.pupil_size()
```

Now place that second `inline_script`  item in `trial_sequence`, as indicated in Figure 4. In its **run phase**, place the following line:

```
exp.set('pupil_size', eyetracker.pupil_size() - start_pupil)
```


The `exp.set` bit is just so that the variable 'pupil_size' is automatically logged by our **logger** item. Pupil_size will be a column in the regular OpenSesame datafile, and we won't have to dive into the more complex eyetracker datafile.

Congratulations! Your first pupillometric experiment is complete. Feel free to add some instructions at the start. Make sure to save your experiment before continuing.

Part II: Analyzing pupillometric data (best done after Thursday)

Imagine having built your pupillometric experiment in a world where you had no other obligations, and a little while later you have collected data with the help of 24 participants. It is now time to inspect that data, with linear mixed-effect models (LMMs) in R.

Throughout Part II, you'll see **Questions** coming up. Please send in your answers for these questions together with the experiment that you created in Part I.

From the Module of Dec 1st, download `pupillometry_data.osexp`. This can be run in OpenSesame, and creates your own personal dataset. Before running it, click on the `create_data` `inline_script`  item and go to the 51st line of code. You'll have to change the directory there into the directory of a folder on your own computer (see Figure 5); but do make sure that it ends with `data.txt`.

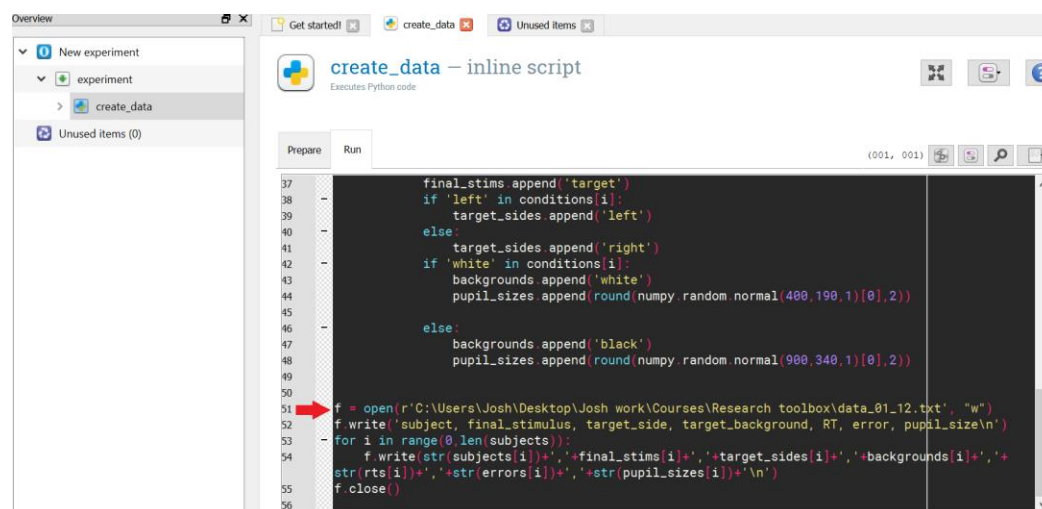
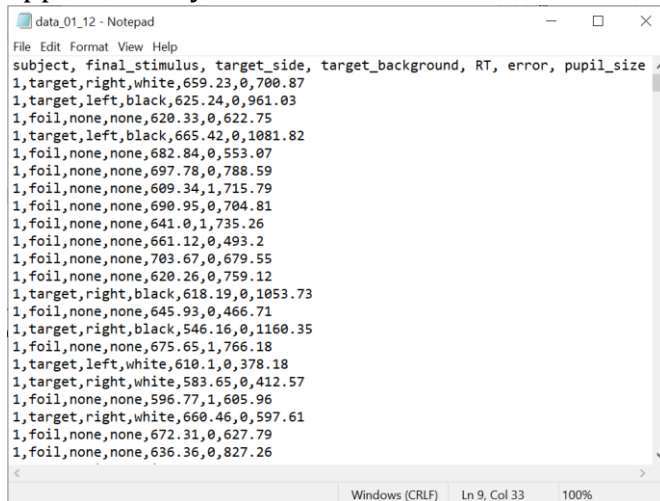


Figure 5. Data simulation script. On line 51, change into your own directory.

When running the ‘experiment’, a textfile will be created on your computer. It will look approximately as follows:



```
data_01_12 - Notepad
File Edit Format View Help
subject, final_stimulus, target_side, target_background, RT, error, pupil_size
1, target, right, white, 659.23, 0, 700.87
1, target, left, black, 625.24, 0, 961.03
1, foil, none, none, 620.33, 0, 622.75
1, target, left, black, 665.42, 0, 1081.82
1, foil, none, none, 682.84, 0, 553.07
1, foil, none, none, 697.78, 0, 788.59
1, foil, none, none, 609.34, 1, 715.79
1, foil, none, none, 690.95, 0, 704.81
1, foil, none, none, 641.0, 1, 735.26
1, foil, none, none, 661.12, 0, 493.2
1, foil, none, none, 703.67, 0, 679.55
1, foil, none, none, 620.26, 0, 759.12
1, target, right, black, 618.19, 0, 1053.73
1, foil, none, none, 645.93, 0, 466.71
1, target, right, black, 546.16, 0, 1160.35
1, foil, none, none, 675.65, 1, 766.18
1, target, left, white, 610.1, 0, 378.18
1, target, right, white, 583.65, 0, 412.57
1, foil, none, none, 596.77, 1, 605.96
1, target, right, white, 660.46, 0, 597.61
1, foil, none, none, 672.31, 0, 627.79
1, foil, none, none, 636.36, 0, 827.26
Windows (CRLF) Ln 9, Col 33 100%
```

Congratulations! You now have some data to work with.

The matrix comprises 7 columns, and a row per trial:

subject: indicates subject number (1 to 24).

final_stimulus: indicates whether the final stimulus was a target or foil (not seen before).

target_side: indicates whether the target was on the left or right (or ‘none’ in case of foil).

target_background: indicates target’s background color (or ‘none’ in case of foil).

RT: response time in ms.

error: indicates whether trial was answered incorrectly (0=no, 1=yes).

pupil_size: pupil size.

2.1 LMMs in R: preparing the script

In the lecture of Thursday 28th, we will work with an analysis script in R. You can download this script from the Module of Sept 28th on Canvas (the file is called *LMM_script.R*), and use it as a template for the present analyses.

Rather than reading in `workshop_data.txt`, make sure to read in `data.txt`. In the line above this, also make sure that the path is specified correctly. At this point you can save the script under a new name, so that the original script isn’t lost due to the upcoming modifications.

The old script filtered the data on trials for which ‘error’ was set to 0. It would be good to this here too – so we can leave that bit of code. We’ll only analyze trials in which the participant correctly memorized a stimulus.

Additionally, we want another filter. Namely, we’re mainly interested in the trials where the target was probed (and we want to see if the target’s background color affected pupil size in these trials). Add the appropriate line of code for this (you need the same syntax as the line that removes incorrectly answered trials).

After this, we see a section of code by which trials with atypical RT values are excluded (to be precise, trials for which $|RT| > 2.5$ SDs from the mean). We can actually leave this as is.

Question 1: We could choose to exclude trials with an atypical pupil size in a similar fashion. Do you think this would be a good idea? Why so, or why not? (*Note, there are no wrong answers per se!*).

After excluding the outliers, the old script shows an explanation for the old dataset that doesn't apply to our dataset. In fact, you can ***delete*** this whole bit:

```
# We may have factors that we want to specify as such. In the matrix, our IV
# 'distractor' has values 0 and 1, for the no-distraction and distraction conditions,
# respectively. Let's just give labels "no" and "yes", respectively:

data$session <- as.factor(data$session)
data$distractor <- factor(data$distractor,
labels=c("no","yes"))

# We can choose reference levels for our factors of interest.
# We can then interpret all effects as differences from the reference

data$distractor <- relevel(data$distractor, ref="no")
data$session <- relevel(data$session, ref="control")

# Here is the code for running a very simple LMM with distractor as fixed effect,
# and by-subject and by-item intercepts as random effect
```

That leaves us with the bit of code that runs the actual LMM. As you may recall with the help of the slides of Nov 24th lecture, we have to specify our Dependent Variable of interest, our fixed effect(s) and our random effect(s).

Question 2: What is our DV, and what are our FE(s) and RE(s)?

Based on your answer for Question 2, modify the model code accordingly. Also include random slopes in addition to the random intercepts.

Question 3: With the inclusion of random slopes, what possibility do we take into account? Make sure that your answer concretely involves the DV indicated in the previous question.

2.2: Analyzing the data

Your code is ready to be run now. Just select everything (cntrl+a on Windows, or ⌘+a on a Mac), and press cntrl+r (or ⌘+r).

Question 4: Please copy (either by means of a print screen or by copying the code manually) the results onto your answer sheet.

Question 5: Please provide a summary of the results (including the *b*-value, SE and *t*-value) and, thus, an answer to the central research question.