Table of Contents

[I. MTurk behavioral data analyses: 2](#_Toc37153445)

[1. Concat\_descriptive\_network\_attack.ipynb 2](#_Toc37153446)

[a) Check for inconsistent naming among tasks and missing of necessary columns. 2](#_Toc37153447)

[b) Generate concat data from raw data 2](#_Toc37153448)

[c) Generate descriptive stats (qa files) from concat data 3](#_Toc37153449)

[2. main\_DVs\_analysis\_and\_corr\_heatmap.ipynb 4](#_Toc37153450)

[a) One table with all SE, DC, and DE variables (using RT or Acc; 3 steps: taskDict, individual table, one concat table) 4](#_Toc37153451)

[b) Heatmap among all SE, DC, and DE results 5](#_Toc37153452)

[c) Task-cluster by Task-cluster correlation matrix 5](#_Toc37153453)

[d) Separate condition data without any subtraction (as with DC or DE) 5](#_Toc37153454)

[e) Check for identical columns in separate-conditions data frames 6](#_Toc37153455)

[3. ezdiff.ipynb 6](#_Toc37153456)

[a) Import data and remove duplicated columns, mean and STD rows, and prefix from colnames 6](#_Toc37153457)

[b) Wagermakers et al.'s (2007) EZ-Diffusion 6](#_Toc37153458)

[II. Shared control 6](#_Toc37153459)

[III. MTurk HIT 7](#_Toc37153460)

[1. Deployment 7](#_Toc37153461)

[2. Contact 7](#_Toc37153462)

# MTurk behavioral data analyses:

More comments in the code. This is a documentation (of general purposes).

|  |  |
| --- | --- |
| Notebook | Main purposes |
| concat\_descriptive\_network\_attack.ipynb  (Documents/Network\_Pilot/) | Process raw data, concatenate processed data |
| Generate QA files (i.e. descriptive stats) |
| main\_DVs\_analysis\_and\_corr\_heatmap.ipynb  (Documents/Network\_Pilot/) | 1 table with all single effects (SE), dual-contexts (DC), dual-effects (DE), using RTs or Accuracy Rate |
| 8x8 table averaging DCs and DEs by task-cluster |
| Heatmap among all SE, DC, and DE results |
| Task-cluster by Task-cluster correlation matrix |
| Separate condition data without any subtraction (as with DC or DE) |
| Heatmap for correlation b/w slowing and effect-change |
| ezdiff.ipynb (Documents/Network\_Pilot/explore\_n33/ezdiff) | EZ-Diffusion model |

## Concat\_descriptive\_network\_attack.ipynb

* raw\_dir='' is where all raw and concat data are for all 66 subjs.
* explore\_dir='' is where data (concat, QA, stats) related to the 33 explore subjs are.
* abbrev={} is a dictionary of abbreviations for DV names, for the purpose of shortening column names when combining multiple DV:condition to make column names.
* explore\_id={} is the list of 33 exploration subjects.

### Check for inconsistent naming among tasks and missing of necessary columns.

* + The JS for some tasks has typos (predictive instead of predictable),
  + Or missing name (directed condition instead of directed forgetting condition),
  + Or no current\_trial (which is useful to exclude unimportant rows during later data-processing).
  + (These will be dealt with in “### Generate concat data from raw data”.)

### Generate concat data from raw data

* + Remove unimportant rows and columns and concatenate each subject’s task data file to one task data file with all subjects’ data for that task in it.
  + for subj\_file in glob(raw\_dir + 'A\*/\*'): A\* is a subject’s folder with /\* as 1 of the 36 task data files completed by that subject
  + # really avoid any malfunctioning correct\_trial: there is one task recording incorrect values for “correct\_trial,” which will be used later to calculate accuracy.
  + for colname in df.columns: change inconsistent naming
  + if 'trial\_id' in df.columns: works for most tasks to keep only “test\_trial” rows
  + processing specific to tasks:
    - predictable task switching: remove 1st trial, using current\_trial>1, or \*\_condition has nan values or string values “na”, depending on what’s available. (This mess is the result of javascripts across predictable task switching tasks are inconsistent.)
    - n-back task: remove first 3 trials
    - cued task switching: remove 1st trial
  + ### check which tasks had malfunctioning correct\_trial (that's automated by javascript): show that said task(s) that recorded “correct\_trial” incorrectly.

### Generate descriptive stats (qa files) from concat data

* + General reading (file path, exp\_id, worker\_id).
  + Generate DVs of interest (done here—i.e., after the processing stage—as concat data should have all original levels of a condition):
    - “H\_congruent” and “F\_congruent” to “congruent” etc.,
    - 7 shape-matching levels to “DISTRACTOR,” “CONTROL,” and the rest.
  + ###prep for generation of colnames of qa\_df:
    - get all columns with “\_condition” in name,
    - and generate new columns in concat data for later calculation of SSRT (specific to stop-signal tasks).
  + The list conditions ranges from 1 to 4, indicating the type of task (e.g., having 1,2,3, or 4 DVs/conditions). The rest of the script is in chunks, each starts with “### Tasks with [n] condition variables.” The following steps are repeated for each of those:
    - Get a list of all conditions of each DV in the task, then sort them alphabetically.
    - Loop through all those conditions (e.g., 1,2,3 for the “delay\_condition” column) to string together DV:condition (e.g., DELAY:1). Next, string together stat\_DV:condition (e.g., m(rt)\_ DELAY:1) and append to list of column names of the QA data frame.
    - Create QA data frame (qa\_df) with workerIDs as index and names above as columns.
    - Loop through all conditions of each DV again to create all DV:condition cases, all stat\_DV:condition cases, then use them to fill later-calculated descriptive stats in the correct columns of the QA data frame (as stat\_DV:condition here is identical to what generated to create qa\_df’s columns). This repeated looping is because the prior loop must be finished for a list of column names to be ready to create the QA data frame. (Surely there’re better ways, but under time pressure, this was what I got that worked.) Then, for each DV:condition case, for each subject, a concat data frame, which has all subjects’ data for one of the 36 tasks, is subset (e.g., df\_subj = concat\_df[…]). This df\_subj will be used to calculate, for all tasks, m(rt), sd(rt), and accuracy rate, and, specific to stop-signal and go-no-go tasks, some other stats such as ssrt, nogofail\_rate. How it runs is: 1) create an object that is stat\_DV:condition; 2) create an object that is the result of, for example, averaging all ‘rt’ where ‘correct\_trial==1’ in the current df\_subj; 3) index qa\_df exactly at row as the current workerID and column as the object from step 1, then fill that cell with the object from step 2. This way, the number of nested loops, unfortunately, increases as the number of DVs increases; for example, n\_back\_with\_cued\_task\_switching will loop through all conditions of “delay\_condition,” “cue\_condition,” and “cue-task\_condition,” which, again, is the task condition in cued task switching task. The repetition and nested loops are not pythonic and dynamic; hopefully, someone with a wider bandwidth and/or less time pressure will improve or rewrite this script.

**Note: 1)** accuracy rate as in the current state of the script is calculated to be used later in an EZ-Diffusion model (i.e., by adding half an error to where accuracy==100%). Read in-script comments for switching back to normal calculation. **2)** correct\_trial is regenerated at top of the process and concat part/cell such that as long as key\_press equals correct\_response, correct\_trial is recorded ‘1’, unlike the original mechanism by which commission in no-go/stop trials was ‘0’ by default. This way, when the loop is in ‘stop’/‘nogo’ condition, counting # of ‘1’ (e.g., acc \_SS:stop) would give the number of responses that would have been recorded as correct had those ‘stop’ trials been ‘go’ trials.

* + ###drop some stats automatically generated that are redundant and DVs not of interest before writing out QA data frames: for example, for now, only ‘neg’ (target in the forget set) and ‘con’ (target in neither the forget set nor the memory set) are of interest, but not ‘pos’ (target in the memory set). Read related comments in #writing: to choose which line.

## main\_DVs\_analysis\_and\_corr\_heatmap.ipynb

### One table with all SE, DC, and DE variables (using RT or Acc; 3 steps: taskDict, individual table, one concat table)

* + ### Create a dictionary: {'taskname': {'dv0':[val0,val1], '...':[...,...]}}
    - Read in concat files. Remember that concat files were generated having all original conditions. Therefore, here again collapse across some levels not of interest (e.g., across H or F).
    - Examples of output in comments in script.
  + ### Table for single effects (SE), dual-contexts (DC), dual-effects (DE), using RTs or Accuracy Rate; one table for each task
    - Read in QA files (36 files for 36 tasks).
    - Subset data for only relevant columns (using RTs or Accuracy)
    - For each type of task (i.e., how many DVs one has), call in the relevant abbreviated DV names, and corresponding conditions/levels, with the first level being the baseline always. Task-specific exceptions are commented in the script. Using filter to subset data for only that condition/level, then subtract to get relevant DC, DE results. Write output.
  + ### one table with all SE, DC, DE data, using RTs or Accuracy
    - Switch input and output directories to use RTs or Accuracy.
    - Read in all individual tables from previous step then concat to one file.
    - Sort out SEs columns so they can be first in each task cluster
    - Sort out DC and DE columns by task cluster
    - Concat then write output.
  + ### 8x8 mean DCs and DEs summary
    - Not primary purpose of this part, just a side look.
    - There are 8 task clusters, each of which could be represented in name by its’ single task. Each cell in this 8x8 matrix will contain the mean of all the DC and DE results between some two tasks.

### Heatmap among all SE, DC, and DE results

* + ### correlations among all SE, DC, and DE data (from 33 explore subjs)
  + ### heatmap: 1's on diagonal replaced by means of corresponding raw data
    - Change input and output directories to use RTs or Accuracy
    - Data set to fill in diagonal: annotData
    - Data for off-diagonal: concat\_mainDVsAcc\_corr\_index=False.csv
    - The color of the heat map is retained such that the highest (reddest) are still diagonals (despite the data used to fill in for annotated values), and the rest having color depending on the correlation coefficient of the pair of variables.

### Task-cluster by Task-cluster correlation matrix

* + #### Each cell has the average correlation of all SE, DC, and DE results of a task cluster with those of another task cluster
    - Input: concat\_mainDVs\_corr from 2. b), first cell, ### correlations among all SE, DC, and DE data
    - Basically, for every 15 rows and 15 columns of concat\_mainDVs\_corr, subset to a data frame, remove the 1’s in the diagonal, average all correlation coefficients, fill in the 8x8 data frame accordingly.
  + #### Plot (heatmap)

### Separate condition data without any subtraction (as with DC or DE)

* + #### Separate conditions, no further subtractions as when calculating for DC or DE results
    - Input: QA files, then filter according to ‘choice’ (see in script).
    - Columns/conditions from QA files, which have not gone through any further calculation (e.g., subtraction to get DC or DE results), are added to separate\_conditions\_dict whose keys are task-cluster names.
    - Go through QA files for single tasks first, then for dual tasks, so that when adding data frames to an existing key (i.e., task cluster name), single-task conditions always appear before dual-task conditions.
    - Choice of keeping all dual-task conditions/columns (see separate\_conditions/accuracy, or separate\_conditions/RTs) or averaging across them (see separate\_conditions/condensed).
    - Choice of calculating slowing and effect-change or not (‘val2’ in script)
  + #### Heatmap for correlation b/w slowing and effect-change (RTs or Accuracy depending on slowing\_vs\_effectChange generated above)

### Check for identical columns in separate-conditions data frames

* + Check if there was an error when generating concat\_separate\_conditions\_[RTs/Accuracy].csv by checking if any two task-cluster data frames are identical.

## ezdiff.ipynb

### Import data and remove duplicated columns, mean and STD rows, and prefix from colnames

* + Input data drawn from both rt and accuracy separate-conditions concat data (output of 2d) as EZ-Diffusion model asks for both rt and accuracy rate.

### Wagermakers et al.'s (2007) EZ-Diffusion

* + ezdiff/ezdiff.py has already had conversions from milliseconds (recorded in all data so far) to seconds (used in model)
  + Choice of writing EZ-Diff output for all conditions for all tasks onto one worksheet, or conditions of each task cluster (1 single, 7 dual) onto one separate worksheet.

# Shared control

* + Documents/analog-test/analog-test/Program.cs is the script to read and show output from the analog keyboard.
  + When in the same folder as Program.cs, run “dotnet run”. (You can try running this .cs file on Visual Studio as well.)
  + wootility-beta (3.4.1) can be used to update the firmware (of the keyboard). An example of when to do this could be when this is shown on terminal:

Hello Analog SDK!

Analog SDK Successfully initialised with 0 devices!

Error getting devices: 0

Read failed with NoDevices

Read failed with NoDevices

Read failed with NoDevices

**Note**: the latest version might not be available to public, but only on the community’s Discord. In that case, when in the Wootility software interface, go to Settings/Update to see any announcements.

* I’ve contacted them at [social@wooting.io](mailto:social@wooting.io). They were responsive, to an extent, last time I reached out.
* I’ve posted issues, and got replies, on this Github: <https://github.com/WootingKb/wooting-analog-sdk>

# MTurk HIT

## Deployment

* + Setting up an MTurk HIT via Expfactory (hereafter, Exp) is different from doing it directly on the MTurk Requester site. The former has been the way to deploy batteries on created on Exp; the latter has been used to set up (dummy) payment HITs. Note that those set up via Exp will not appear on the HIT management page on MTurk Requester.
  + Either way, make sure you’ve purchased prepaid HIT credits on the Requester site before trying to deploy a HIT. <https://requester.mturk.com/> -> My Account -> Purchase Prepaid HITs -> …
  + If you want to deploy an existing battery on Exp, you’ll need to ask Ross to give you creator access to that battery. Otherwise, you can only edit those you create yourself. Ask Ross also for log in credentials. Once you’ve been logged in:
    - go to the battery you want to deploy,
    - Serve Battery -> New HIT,
    - Deployment to Amazon…: choose “Amazon Mechanical Turk”.
    - Reward: should put half of total payment to be this automatically paid amount (upon a worker’s submission of his work); the other half should be paid through dummy HIT (as a safe guard for cases that did bad work and only wanted the money).
    - v Lifetime in hours: 7 days is one option.
    - V Assignment duration in hours\*: last time, we gave them 7 days.
    - V Max assignments: sample size.
    - Auto approval delay in seconds: 3 days, as last set up.
    - Worker qualification custom: <https://requester.mturk.com/> (log in info in GroupAccount document in Lab Procedure) -> Manage -> Qualification Types -> choose one from ID.
    - Operator to compare variable to value: equals to 1 normally means to include those granted the ID from “Worker qualification custom.”
    - Examples for other settings can be found by looking at previous HIT set up (e.g., Network Attack (post pilot) -> HITS -> Network Attack 26 (click on “Details” under the title)).
    - Then click “Save”. On the same battery, if a new HIT appears on the “HITS” tab, you succeeded. If not, reach out to Ross for assistance.

## Contact

* + Just be patient with the MTurkers. Those who give you real anxiety should be put on the Blacklist document. They will most likely contact [selfreg2016@gmail.com](mailto:selfreg2016@gmail.com) rather than the Poldracklab email. Log-in credentials can be found in the GroupAccount doc in Lab Procedure.
  + NotifyWorkers.py (in Documents/MTurk/) has examples of using AWS (which uses boto3) to do many procedures related to MTurk (approve assignment, send bonus, get HIT status etc.).