The Swedish National Core Facility for Magnetoencephalography (NatMEG)

Presentation code template

Auditory oddball experiment

Author

Judith Recober

January 16, 2025

Head of the core facility

Christoph Pfeiffer

1 Introduction

This code example is very similar to the other. However, here we introduce 3 new things:

- new stimulus: sound. Notice that this stimulus is not defined in the SCE code because we get the sound from the sound files (you can play them)
- we add a practice block, which is not interesting to record, but it is needed to be sure that the participant has understood the task
- in the previous one, there was a resting state where nothing was happening meanwhile. In this case though, in each block (there are 2) different sounds are played. Thus, we have a .txt file where each row corresponds to a trial. A trial is basically a group of sounds that will be played and have the same tone and at the end of the trial, a sound with a lower or higher tone will be played. Thus, we store the information of this .txt file in an array variable. Thus, in this case, the length of your experiment is determined by the length of the .txt file and the duration of each sound per trial.

Notice that here we will use the same subroutines functions and the same stimulus defined in the SCE code.

2 Main PCL file

2.1 Initial parameters

You need to think about what parameters are important for the interface of your experiment.

- First of all, if you want to show any text instructions you may want to have different language options
- In the case of this experiment, we also added a block for the participant to practice first. However, since the experiment was carried out 2 times per participant, in the second time it was not necessary to do the practice exercise. Think if in your experiment code there are blocks for which you need dinamic changes (go over or not go over this block for example) across different experiments with the same or different subjects. If that is the case you need parameters to control tese changes more easily. For example, in the case of this code, instead of comenting the practice exercise block every time we run it (which would be not very practical neither time-efficient), we have a parameter that we include in a conditional statement to decide if we want to execute that part of the code or not.
- Ports
- Subroutines

```
###################### PARAMETER THAT CAN BE CHANGED BETWEEN EXPERIMENTS ######################
# Add parameter that you may want to change between experiments, so that you get easy access to them
string language = "eng"; #language can be "eng" or "sv"
string PracticeYes = "y";  #skipping practice block "y" or not skipping "n"
```

```
##################### FILES THAT NEED TO BE INLCUDED ######################
# First of all, we need to include all the necessary files that will be used in this experiment
include "subroutines.pcl";"

begin; #beggining of our experiment

##################### PORTS DEFINITIONS ######################
# in the app to manage the exeriments, there are output ports. We need to know this
# so that the output of the code goes to the correct output port of the system
int AFport = 2;
int myMEGport = 1;
output_port audiofile = output_port_manager.get_port(AFport);
output_port MEGport = output_port_manager.get_port(myMEGport);
output_port OPMport = output_port_manager.get_port(3);


default.present(); #this function presents the settings in the screen.
parameter_window.remove_all(); #This function removes any previously configured parameters
```

2.2 Prepare your experiment

In order to start running the experiments, you may need to define different variables like strings or lists or counters.

Let's analyze this experiment: This is an auditory oddball experiment were basically some sounds are reproduced. Most of these sounds have a certain "tone" and some specific ones have lower or higher tones in comparison with the standard tone (the one that the majority of the sounds have). So in this experiment, each trial will be something like: beep, beep, beep, beep-higher.

For this experiment we have a .txt file with 3 colums and 60 rows (which is the total number of trials we run per block in a single experiment). In the first colum we define the trial number (1 to 60), in the second colum the number of normal balls before the lower or higher oddball is reproduced and in the third colum a number that can be either "low" (lower tone) or "high" (higher tone). This experiment has 2 blocks, one where the go oddball is high (so the participant needs to press the button when they hear the higher tones) and the no-go oddball is low, and then in the second block is the other way around.

Since the participant may not fully understand the task from our explanation, we also have a practice block (no recording needed for that).

Thus, in this case we need to define the following variables:

- PracticeRunTrials where we will store the definition of each trial in the practice section by using our .txt file "NatMEG-Auditory-trial-practicerun"
- EXPtrials where we will store the definition of each trial in the experiment section by using our .txt file "NatMEG-Auditory-trials"
- Then we define the frequency of the sounds we will reproduce based on their category (normal ball, low oddball or high oddball). We have an audio file where each number represents different sounds with different frequencies.
- Also, when you record your data and you know that something is happening, you may want to set some triggers to actually know when those things are happening. Imagine you show an image to the participant during 3 seconds at some point in your experiment. You will probably record more data at once, not just the 3 seconds. Thus, along the time frame of your recorded data, you may want to know when the image appeared and the image was gone (to for example further analyze that particular part of the data or to know that if your data looks different in that time frame its due to the image that you were showing to the participant). For that, we use triggers. This triggers will be send to the recording MEG and OPM systems. We will further analyze this in other parts of the code.
- Also, if you have dynamic variables that will change during the code (they are not fixed) in if conditions, it is better to first define and initialize them, because otherwise the code will give an error.

```
int trialBlockLen = 6;  #Total number of trials per block.
#This is the total number of  rows in our file (each containing 1 oddball).
input_file PracticeRunFile = new input_file; # prepare out input file
PracticeRunFile.open("NatMEG_Auditory_trial_practicerun.txt"); # define the input file

#Reading the file where the trials are defined and storing those values in a variable

#since PracticeRunTrials is an array and arrays need to have the same type of data,
#we need to devide them into 2 different arrays (one with integers and one with strings)
array<int> PracticeRunTrials[trialBlockLen][2]; # make an array with trialblocklen rows and 2 columns
array<string> PracticeRunTrials2[trialBlockLen][1];  # make an array with trialblocklen rows and 1 column
loop int r=1 until r> trialBlockLen begin
 PracticeRunTrials[r][1] = PracticeRunFile.get_int(); # read all the columns from the practice file
 PracticeRunTrials[r][2] = PracticeRunFile.get_int();
 PracticeRunTrials2[r][1] = PracticeRunFile.get_string();
 r=r+1;
end;
PracticeRunFile.close(); #close file
```

```
#Now the same but with the experiment section

int blockLen = 60; #Total number of trials per block.
#This is the total number of  rows in our file (each containing 1 oddball).
input_file myfile = new input_file;
myfile.open("NatMEG_Auditory_trials.txt");  #opening the file where the 60 trials are defined

array<int> alltrials[blockLen][2]; #alltrials will store all the info of the 60 trials are defined
array<string> alltrials2[blockLen][1];
myfile.open(fname);
loop int r=1 until r> blockLen begin
 alltrials[r][1] = myfile.get_int();
 alltrials[r][2] = myfile.get_int();
 alltrials2[r][1] = myfile.get_string();
 r=r+1;
end;
myfile.close(); #closing the file



#we have an aodio file where each number repesents different sounds with different frquencies
#SoundNum variable will tell us which sound should be played for the normal beep, low and high oddball

int NormalBeepSoundNum = 37; # sound number 37=39_B4 -> 987 Hz
int LowOddballBeepSoundNum = 75; #1; sound number 75= 43_G4 -> 782 Hz
int HighOddballBeepSoundNum = 79; #2; sound number 79= 43_D5 -> 1173 Hz

#Also, for post-processing of data, we need to know when a normal or a high or a low odball happend.
#For this, we label the sound with a number (called trigger) to send it to the system so that
#afterwards we can identify when the stimulus occured

int NormalBeepTrigger = 1; #trigger code for normal beep (1 for sound onset)
int OddballBeepTrigger = 3; #2#trigger code for no-go oddball (2 for no-go, 1 for sound onset)
int GoOddballBeepTrigger = 5; # trigger code for go oddball (4 for go, 1 for sound onset)
int beepTrigger = 0;

int ITI = 1250; #1250 ITI ms
```

```
#some variables that need to have a defoult inicialization before using them
int WhichOddball = 1; # Initial identification of the go or not go oddballs
#1: NoGoOddball or 2: GoOddball
int beepTrigger = 0;
int soundNum = 0;



array<string> GoOddBallFrequency[2][2];  # create empty array with two rows and two col
GoOddBallFrequency[1][1] = "high";  #assing values to the empty array
GoOddBallFrequency[2][1] = "low";

if language == "sv" then
 GoOddBallFrequency[1][2] = "högre";  #assing values to the empty array in swedish in column 2
 GoOddBallFrequency[2][2] = "lägre";
else
 GoOddBallFrequency[1][2] = "higher";  #assing values to the empty array in swedish in column 2
 GoOddBallFrequency[2][2] = "lower";
end;

# This line is used to suffle the orther when deciding the GO ODBALL of the block.
# so there si a 50% change that the first GO oddball is high  and 50% of being low
GoOddBallFrequency.shuffle();  # randomize order of High and Low as the go-oddball
```

2.3 Practice section for the participant

In every practice exercise or experiment we have 2 loops:

- The first loop will correspond to the block number. Every exercise or experiment will have 2 blocks: 1 block for the high oddball and 1 block or the low oddball (the order of these 2 blocks is random).
- The second loop will correspond to the trial number inside 1 block. As explained before, the practice exercise has 6 trials per block and the experiment has 60 trials per block.

Since this is just a practice exercise, we are not using the triggers for the MEGport or OPMport. The general structure of this practice exercise is the following:

1. loop through each block
   (a) Show instructions to the participant explaining which is the go oddball in this block (high or low) in the

correspoding language

(b) Shuffle the order of the tirals

(c) Present the fixation cross

(d) loop through each trial

i. Take the oddball type for that corresponding trial (this is the WhichOddball variable): low or high

ii. Take the total number of normal beeps that the participant will hear before the oddball in that corresponding trial

iii. (Meanwhile show the progress in our screen)

iv. compare the current oddball type with the go odball in the block to know if the current oddball is a "go" or "no-go"

v. loop to play the normal beeps (determined by the previous step)

A. Get the sound from the audio port (with the corresponding frequency)

B. wait interval

C. Play/trigger the sound

D. Wait interval

vi. After the normal beeps, play the corresponding oddball sound

```
if PracticeYes == "y" then
 loop int PracticeBlockCounts = 1; until PracticeBlockCounts > 2 begin; #loop per each block

   # INSTRUCTIONS: define and show which is the GO odball of this block
   if language == "sv" then
    ShowInstructions("En kort övning innan experimentet startar \n\n",
                "Tryck på knappen när du hör den \n\n ",
                GoOddBallFrequency[PracticeBlockCounts][2] + " tonen");
   else
    ShowInstructions("A short exercise before the experiment starts \n\n",
                "Press the button when you hear the \n\n ",
                GoOddBallFrequency[PracticeBlockCounts][2] + " tone");
   end;

   PracticeRunTrials.shuffle(); #shuffle the order

   p_Fixation.present();# present fixation cross
```

```
loop int MyCount=1; until MyCount > trialBlockLen begin #loop per each trial, 60 in total.
        #MyCount controls the trial number

 WhichOddball = PracticeRunTrials2[MyCount][1]; # take the current oddball
 int TotalNormalBeeps = PracticeRunTrials[MyCount][2]; #total number of normal beeps before oddball

        #if the oddball of this trial is of the same type as the go oddball of the block,
        #then it is a go oddball
 if WhichOddball == GoOddBallFrequency[PracticeBlockCounts][1] then
  ball_textTRIAL = "go";
 else # else, it is a no-go odball
  ball_textTRIAL = "no-go";
 end;

 # show progress in our screen
 ShowAltText("Block: " + string(PracticeBlockCounts) + "  the go-oddball is " +
             GoOddBallFrequency[PracticeBlockCounts][1], "n of OddBalls: " + string(MyCount) + "/" +
             string(trialBlockLen), "Odball type: " + ball_textTRIAL);

 # Present normal beeps
 loop int normalBeeps=1; until normalBeeps > (TotalNormalBeeps) begin

  audiofile.send_code(NormalBeepSoundNum,80); #load sound
  wait_interval(50); #wait interval
  audiofile.send_code(128,10); #trigger sound
  wait_interval(ITI); # wait interval
  normalBeeps = normalBeeps + 1
 end;

 #sound that needs to be played for the odball knowing if it is a low or high oddball
 if WhichOddball == "low" then  #low oddball
  soundNum = LowOddballBeepSoundNum;
 else #high odball
  soundNum = HighOddballBeepSoundNum;
 end;

 audiofile.send_code(soundNum,80); #load sound
 wait_interval(50); # wait interval
```

```
    audiofile.send_code(128,10); #trigger sound
    wait_interval(ITI); # wait interval

    #Update count
    MyCount = MyCount+1;

   end; #trial loop
   PracticeBlockCounts = PracticeBlockCounts + 1;
  end;
end;
```

2.4 Main experiment loop

This code follows exactly the same structure, however this time there are more trials per block and each time we play a sound, we also send a trigger to the MEGport and the OPMport so that when we analyse our data we can know when each sound was played.

```
########################## START EXPERIMENT ##############################################
loop int block=1; until block > 2 begin;

 # suffle all the data
 alltrials.shuffle();
 alltrials2.shuffle();

 # ask sleppiness to the participant
 kss_routine(language);

 #show instructions and define the GO ODDBALL (high or low).
    #Remember that GoOddBallFrequency was also shuffled before
 if language == "sv" then
  ShowInstructions("Tryck på knappen när du hör den \n\n ", GoOddBallFrequency[block][2]
        + " tonen", " ");
 else
  ShowInstructions("Press the button when you hear the\n\n ", GoOddBallFrequency[block][2]
        + " tone", " ");
 end;

 p_Fixation.present(); #present fixation cross
```

```
# loop per each trial
loop int MyCount=1; until MyCount > blockLen begin

 WhichOddball = alltrials2[MyCount][1]; #take the oddball of this trial
 int TotalNormalBeeps = alltrials[MyCount][2]; #number of normal beeps before the odball in this trial

 if WhichOddball == GoOddBallFrequency[block][1] then #check if it is a go or no-go odball
  ball_text = "go";
  beepTrigger = GoOddballBeepTrigger;
 else
  ball_text = "no-go";
  beepTrigger = OddballBeepTrigger;
 end;

 #show progress in our screen
 ShowAltText("Block: " + string(block) + "  the go-oddball is " + GoOddBallFrequency[block][1],
       "n of OddBalls: " + string(MyCount) + "/" + string(blockLen), "Odball type: " + ball_text);

 # Present normal beepp
 loop int normalBeeps=1; until normalBeeps > (TotalNormalBeeps) begin

  audiofile.send_code(NormalBeepSoundNum,80); #load sound with normal beep sound num
  wait_interval(50);
  MEGport.send_code(NormalBeepTrigger,40); # send predefined trigger
  OPMport.send_code(NormalBeepTrigger,40); # send predefined trigger
  audiofile.send_code(128,10); #trigger sound (play it)

  wait_interval(ITI); #Not random ITI
  normalBeeps = normalBeeps + 1
 end;

 # Present oddball
 if WhichOddball == "low" then  #low oddball
  soundNum = LowOddballBeepSoundNum;
  beepTrigger = beepTrigger + 0;
 else #high odball
  beepTrigger = beepTrigger + OddBallFrequencyHighTrigger;
           # if the go-oddball is low, then the no-go oddball is high
```

```
   soundNum = HighOddballBeepSoundNum;
  end;

  audiofile.send_code(soundNum,80); #load sound
  wait_interval(50);
  MEGport.send_code(beepTrigger,40);   # send our updated beepTrigger
  OPMport.send_code(beepTrigger,40); # send predefined trigger
  audiofile.send_code(128,10); #trigger sound
  wait_interval(ITI); #Not random ITI

  #Update count
  MyCount = MyCount+1;

 end; #trial loop
 block = block + 1;

end; # block loop
kss_routine(language);

endOfTest.present();
end;
```