­­­The goal of this document is to help you understand how your experiment works. By reading this document and also reading the documentation for specific functions you should have an excellent of grasp of your code. I want you to be able to intelligently explain how the experiment was programmed as well as learn a little something about coding. Parts of this document will not make sense until you read further on. I suggest at least two quick read-throughs. I will only cover the pseudoExp and printData since the other function are fairly simple. If you want me to write about those as well, let me know.

I am well aware that much of this document will be common knowledge to you. You did code this experiment after all. If I’m breaking things down to an extremely basic level don’t take it as an insult. I just want to be as through as possible so it is easier for you to learn. Also this is not the most elegant code. There are probably lines that can be removed or changed to be more efficient. It works though and time is an issue.

This document is a learning tool and should be treated as such and you should make an effort to carefully read it. Or don’t; it’s your phd so do whatever the hell you want.

Legend:

Line

Commented out Line

Variable

Function/script specific to your experiment

Function general to matlab

**PseudoExp.m**

*Line 3 -9: Setting rootpath*

In this set of code not much happens. The most important thing is clearing the workspace. This a good habit to have whenever coding a script in MATLAB. You might miss some crap in your workspace from something else which can end up causing a headache later. rootPath is just the current directory which can be gotten from the pwd command (print working directory). You may have used the bash version of pwd while navigating the terminal on a linux or mac system.

Now checkout line 9. It uses a very useful function called assert. This function throws an error if a given condition is true. So in this case we are checking if the RUN.EXP file exists (using the aptly named exists function) and if it doesn’t we must be in the wrong directory. This is Carlo’s code and I’m not sure what the point of it is but it is a good chance to illustrate the use of assert.

*Line 12-75: Taking in experiment parameters*

This chunk of code doesn’t need much explanation. You wrote most of it and you should know what it does. I made a few small changes like the practice vs practise thing. I also changed the first argument instruction to just TMS On/Off instead of the demo versus experiment mode which I found confusing. I did add a flag to signal if we are resuming or starting a new run.

*Line 78-86: Setting the button responses*

Not much to say. Once you have the button box this where you will have to change the values of realButton and pseudoButton.

*Line 88-90: Useless code I think*

Yeah I’m not sure why this is here. I never use the variables I set here. Or at least I think I don’t. Leaving it in because I don’t want to break something by removing it and because I’m lazy.

*Line 97-119: Buffering sounds*

This part of the code can be broken down into three section, one to buffer each sound. There are 3 sounds in this experiment, feedback indicting a right response, feedback indicating a wrong response, and a sound to mimic the discharge of the coil. All three sounds were modeled and recording into a .wav file before hand.

The first section buffers the correct response indicator. Line 97-98 read the .wav file for the correct sound. The function audioreader is how we read the .wav file. Calling to this function returns two things: y which is the audio data itself and freq which is the sample rate. Now for reasons unknown to me we have to transpose the audio data y (Line 99) in order to set the number of channels or some shit like that (Line 100). To be frank I don’t understand all the technical in setting up the sound drivers optimally but whatever it works. We then initialize the psychtoolbox sound drivers inputting 1 as an argument so that it enters low latency mode (Line 100). Line 101 uses PsychPortAudio to set up the parameters for the actual sound. The “Open” argument is what signals that we are setting up parameters. I can’t remember what all the arguments are (you can google this if you care) but the important ones are the 1 (number of repetitions of the sound) and nrchannels (number of sound channels to use). Once that is done we call to PsychPortAudio this time with the “Buffer” flag (Line 103), signaling that we are loading the sound (into the buffer for faster playback. Same process is repeated with the other two sounds.

If it isn’t obvious enough, I don’t quite understand this code mostly because I know jack shit about anything related to audio. Hopefully I explained it enough that you have a general idea what I did and can just google the details if ever need to use this again.

*Line 127-137: Initializing Settings 1*

This batch of code initializes settings for your experiment such the mask duration and stim duration. Unchanged from previous code

*Line 142-159: Starting Datapixx*

As far as I can tell the is for the TMS portion of your experiment. Nothing I touched.

*Line 161-163 Initializing Settings 2*

Line 161 calls to a function checkArgs. All this does is take in the args and puts them in a standard form. For example, if someone puts male instead of m these will change male to m. Not really useful to be honest. This is a form of error handling that is nice on a consumer product but kind of meaningless here. Bear in mind I also may have chose to do things differently than Carlo envisioned.

Line 162 calls to setParam. This is a very important function. This sets the fixation duration and number of trials in you experiment. **CURRENTLY THIS FUNCTION USES MY TEST VALUES! YOU NEED TO CHANGE THIS!** You can experiment here in order to get the timings downs. This function requires that you give it the phase, block. I had to subtract some time for the fixation duration in the practice phase because of the sound. Oddly enough didn’t have to do this with the other condition but you should experiment with the timings to get it perfect.

Line 163 creates a Boolean if the experiment is in test phase

*Line 167-174: Setting the screens and Keyboard*

Standard stuff to draw to an external key load mexfile. I added a KBCheck so you can deciede when you are ready to show the first screen of the experiment.

*Line 177-181: Creating empty trials*

In this for loop we are creating an empty instance of each trial. Obviously we do not have the information needed to fill them yet. Creating a dummy structure is good form because it allows us to pre-allocate our array

*Line 186-208: Setting File paths*

Line 188 sets the file paths for the stimuli and legends. It does this by calling to the function setPaths. The input for setPaths is block, phase, session and subcode. All these variables contain the information we need to create the file paths to the legends and stimuli for this run and participant.

Line 189-193 set the file paths to record the complete results, stimuli with consistent responses, stimuli with inconsistent or no responses, and a list of stimuli completed which incudes some trial information (this not meant to be human readable but instead is only used for the resume function. Line 192 is commented out because it was used when I was randomizing the legends in the experiment. The purpose of this line was set up a file to record the order of the stimuli after the randomization. This way if you needed to resume stimuli could be reordered into what they were before. This is now pointless because you will be randomizing the legends which is the way it should have been done in the first place.

Line 196-198 gets the current date and time. This is done to tag backup files. The date and time is not human readable so if you want a specific backup file you will have to go into finder and sort by date modified to. The date tag though makes sure that the backups all have unique filenames so nothing is over written in those folders.

Line 201-202 file paths for the back ups. Line 206-210 checks if this is a test phase run. If it is then it gets the path for the file that contains the list of consistent stimuli.

*Line 213-233: Setting file permissions when resuming*

Now that we have set the appropriate when need to create the file identifiers (fid) using fopen and give them the appropriate permissions (ie. read, write, append etc). Now the permissions will change based on whether we are resuming or not. So this if statement checks are we resuming. Let’s look at the first case when we are resuming a previously interrupted run.

Line 215 creates the fid for the results (fid). Now since we are resuming the results file will contain all of the subject information we recorded when starting the run that was interrupted. It will contain the complete trial information for the trials that were completed. Since the run was interrupted the trial information was never run through print data so the last line of the results file is actually the last trial. So as follow we want append the rest of the trials as they occur. That is why we give it the “a” flag telling matlab to append to the current file. So now when we write to fid it will start from the bottom of the file.

Line 216 reads the original order of the stimuli. Read the section *Line 186-208: Setting File paths* as to why it is commented out.

Line 217 creates the fid for the listing the completed stims (fidCompleted). For now, we will set this to read only (“r”). That is because in the next line we want to create a copy of the completed stims into the back up so they are starting at the same point. We then create the fid for the of list completed stims back up (fidCompletedBackUp) giving it the permission to append.

Line 220 create the fid for the backup of the results (fidDataBackup). In this case we just create a new file so you can see the results between interruption easily. We give it the “w” flag to signify a new file (or overwrite a file with this same path). Next chunk of code just writes the participant information into the results back up. Unlike the results it uses the information you entered when starting up the experiment after were interrupted. **YOU MUST MAKE SURE YOU INPUT THE SAME INFORMATION AS BEFORE, IN PARTICULAR THE LEFT OR RIGHT RESPONSE KEY THING**.

*Line 233-261: Setting permissions when starting a new run*

When we start a new run we need to create all new files. So we make the same fid variables except all of them have the “w” flag. We also have to write the participant information into both the back up and results file.

*Line 263-265: Reading the legends*

Line 263 only exists because Carlo for got to add “.txt” to the legpath in the setPaths function. The next line just checks the legend actually exists then it creates the fid for the legend (metaData).

*Line 268-279: Preparing the Stimuli*

Now that we have read in all the information we need for the experiment we need to prepare the trials. We want to create an array called stimuli. In stimuli there are as many elements as there are trials. Each element in stimuli is the stimulus that will be shown at the indexed trial (stimuli(1) will be the stimulus for trial 1, stimulus(2) will be the stimulus for trial 2)

Now first we want to create a stimulus object. An object contains information that is common to a group of a thing. Take a car for example. All cars have 4 wheels, engine, body etc. Thing is not all cars are the same, so the engine in a Porsche will not be the same as an engine in the civic. So what I am trying to get at is all cars have the same types of parts but which specific parts they have identify the car. An object works in similar manner. It has a list fields (characteristics) and values in those fields make up a specific object.

To create our stimulus object we first need to get what values we are going to give it. To do that we first read the line from the legend (starting from the first line). We also need the phase and block along with the information from the legend. Now in order to create a stimulus object we need to feed all of the previously mentioned information separated by commas. The legend information separates each piece of information with commas so we don’t have to change anything there. We do however have to concatenate the phase and block along with the comma separators in the appropriate spot. This is what line 273 does. The structure of the string variable thisMeta will look like this (block), (phase), (legend info: stim ID), (legend info: real or pseudo), (legend info: novel, study or ortho), (legend info: orthographic neighbor, if the word is in the ortho category).

We feed thismeta into the pseudoStim function and we get a stimulus object that corresponds to what was in thisMeta. Stim is the current stimulus that we are initializing (line 274). Line 276-280 assigns correctKey and in line 281 we add stim to the array called stimuli.

*Line 288-305: Setting the display count of each stimuli*

The goal of this block of code is to set the display count for each stimulus. The display count is stored in the stimCount field of each stimulus object. Now the way I have chosen to do this is to take a stimulus and compare it’s stimID to that of the stimuli that appear before it. If the stimID matches another stimID before it we set the current stimulus’ stimCount to 2 otherwise we set it to 1. So to state an obvious fact, in the practice and test phase all the stimuli will have a stimCount of 1, whereas the study phase will have half with the stimCount of 1 and half with stimCount of 2. By default stimCount is set to 1 for every stimulus object.

Let’s look at the specifics of the code. We have an array called stimuli. This contains all of the stimulus objects in the order that they will be displayed. What we want to do is compare each element of the array to every element that is indexed before it. So take stimuli(3) we only want to compare it’s stimID to that of stimuli(1) and stimuli(2). For stimuli(4) we compare it to stimuli(1), stimuli(2) and stimuli(3). The best way to do this within my knowledge is to use two nested for loops. Loop 1 will pick stimulus(i) while loop 2 will go iterate through all stimuli that are indexed before i.

Line 288 sets up the first for loop. This for loop sets i from 2 to the number elements in stimuli (this is obtained using the function numel), and i is used call to a specific index in stimuli. We are starting from the second stimulus because we want to compare it to all the previous stimuli. If we start from the first stimuli there is nothing to compare it to.

Before the second loop is initialized, we create a boolean called repeat (Line 290). The purpose of this variable is to tell us whether a stimulus has been repeated for when we have to set the stimCount. Each time a new value of i is iterated it is set to false.

Now we set up the second for loop (Line 291). This loop will set the value of j from 1 to (i – 1). This will iterate through each stimulus in stimuli up until stimuli(i). It is in this loop we make comparison between stimuli(j) and stimuli(i). We can use an if statement (line 293) to check if the stimIDs match and if they do set repeat to true.

Once loop 2 terminates we check the value of repeat (line 300). If it is false we set stimuli(i).stimCount to 1 (line 303). If repeat is true we set stimuli(i).stimCount to (line 301)

Let’s say we are at the first iteration of Loop 1. That would mean i = 2 and repeat is set to false before any comparisons are made. We then enter Loop 2. We start off with j = 1. We take stimuli(1).stimID and compare it to stimuli(2).stimID. Let’s say they do not match so we do nothing. The second loop is actually completed because the loop only goes to i – 1 which in this case would be 2 – 1 = 1. We check repeat find that is false and set stimuli(2).stimCount to 1. The program then moves into the next iteration of i which is i = 3. Again set repeat to false and then start the second loop. The first value of j = 1. We compare stimuli(1).stimID to stimuli(3).stimID. Again let’s say there is no match we move onto the next iteration of j which is j = 2. We compare stimuli(2).stimID to stimuli(3).stimID. We find that these match so repeat is now true. NOTE: I realize now I should probably put a break at this point so loop 2 doesn’t continue once a repeat is found. It is poor form but not a big deal. Moving on once loop 2 is finished we check repeat and since it is true we set stimulus(3).stimCount to 2.

*Line 308-338: Determining which trial to start from*

First of all of the commented out code here relates to randomizing the stimuli and figuring out the order on a resume. It is no longer needed.

The next thing we need to do is determine which trial to start with. First we check if we are resuming (Line 308). If we are not resuming we must start from trial 1 (Line 313), pretty obvious. Now if we are resuming we need to read the completed stims file which is identified by fidCompleted. I used the function textscan which reads fidCompleted and creates an array (which is stored in getCompleted) where every element is a line from fidCompleted (Line 331-332). So getCompleted(1) is the first line of fidCompleted. Now a simple call to numel will give us the total number of completed trials and we add one to get the correct starting point (Line 334). Finally we close fidCompleted and open it again with the “a” so that we can continue to append trial information to it (Line 335-337).

*Line 341-351: Loading Images*

You asked to have the images loaded outside the actual trials, so I did that. What we want to do is iterate through all of the stimuli, get the file path for each stimulus and load it into a variable and store that in an array. Line 343 gets a stimulus from stimuli. In line 345 we set the file path to the stimulus and then in line 349 we read the image using imread and eventually store it in stimulusImage (with img acting as an intermediary). When we need to show the images we will call to stimulusImage. If could redo this I would add another field to pseudoStim for the preloaded image. This will work though but not the cleanest way to do it.

*Line 353-390: Bunch of Psych Toolbox initializations*

Not going to say much here. I copied most of this from your code. Only line 390 is different because I needed to remove some extra time in fixation due to sounds (I think at least)

*Line 393-488: Running the Experiment*

A lot of this code should look familiar to you. Line 395-397 creates a trial object using pseudoStim. First we pick the appropriate stimulus from stimuli (line 395), then we use that to create a trial (line 396).

Line 400-410 shows the stimulus and activates the TMS if necessary. 414-416 plays the discharge sound of the TMS coil. This only happens when in the non TMS study and test phase. The sound is played using PsychPortAudio. Line 419 uses the delay function to wait 150 ms before looking for key strokes. Matlab has the pause function but for some reason it was causing issues with the onsets so I wrote my own.

Lines 420-464 are fairly self explanatory. Basically it is the same as you wrote before. We check for a response while the stimulus is up. Then we show the mask. If the participant responded before we do not check for key strokes anymore. We do the same thing during the fixation portion. After that if we did not get a response we set responseTime to 0 and if we did get a response we calculate the reaction time (Line 456-461). Finally we calculate the onset of the stimulus (464) Not more much to say here. You wrote most of this.

467-480 records all of the trial results into our trial object. After that it places the trial in completedTrials. The field names should tell you exactly what each value means but if there is any confusion you can ask me. Lines 483-491 play a feedback beep if the participant is in the practice mode.

Lines 495-505 print to the appropriate places. 495-497 print trial information to the console. I formatted in the way I felt was the easiest to read but if you want it changed let me know. 500-501 print the trial info to the results file and the backup, 504-505 does the same with completed stims file. Finally, 507 sets keyFound back to 0 in preparation for the next trial.

*Line 512-540: Fill in trial information on a Resume*

When we resume we want to be able to analyze the trials with printData. We can fill in the previous trials information using what was recorded in the completed stims file. Lines 514-519 read fidCompleted and create an array where each element is a line from fidCompleted (see *Line 308-338: Determining which trial to start from* for more information on this process). This array is called getCT

After that we enter a for loop that reads every element in getCT. This string has all the information we need, except it is tab delimited. So we split the string between tabs and get a new array, data. After that we just feed in the correct information and place the trial in completedTrials.

printData.m

This function takes in the completed trials, phase, file path for the results file, file path for the consistent stim, file path for the inconsistent stim, file path for the consistent stims from study phase (only used if in test phase), the real response button and finally the pseudo response button.

*Line 1 – 157: Initializations*

The first 150 ish lines are just declaring variables. There is a variable for every combination of stimulus type and category for every analysis scenario (all trials, consistent trials and correct trials).

*Line 160 -236: Setting up consistent and inconsistent stims in study phase*

Before we analyze the trials we need to know which stimuli had consistent responses. Line 163 and 164 set up the file we are going to write the stimulus information to. Now we must check the stimuli against each other. There are probably multiple ways to do this and I have chosen a fairly inefficient algorithm but for our purposes it will work with no negative repercussions (overall there are many ways I could have been more efficient in doing this task but yeah whatever) . What I chose to do was take a trial and compare it to all the other trial until I find one with a matching stimID. I then make a decision on whether it is consistent or not and write it the appropriate file. If this stimID has already been written into the file, I do not write it again. Remember we are checking each trial against all other trials so each stimulus will come up twice but we only need to record the result once.

In line 167-168 I set up a nested for loop. The first loop (with the variable i) is the current trial I am working with, and j iterates through all other trials. In line 170 I check to make sure that both i and j are not equal because that would mean I would be referencing the same trial. If they are equal, the program skips to the next value of j. Next step is to check whether trials(i) and trials(j) have the same stimID ( line 173). If they do not match we continue to the next value of j. Now if they do have the same stimID we have to make a decision on the stimulus. First thing I do is check if trial(i) actually has a response (line 176). If there is no response we jump to line 222-226 and write it into the inconsistent response file along with a no response tag (if the stimulus has not already been written). If there was a response in trial(i) we now check it against trial(j)’s response (line 180). If they are the same we now to do two things: first we need to see how the participant was responding, if they were consistently responding pseudo to a real word we will now treat that stimulus as pseudo (Line 186-191). Next we put the trial in the trialsAnalyzed array which is a collection of trials that had consistent responses across all trials of their stimuli (Line 195). Next we want to store the stimulus information in array to write to a file later (stimsC). First we need to check if we haven’t already done that and if we have not add it to the array (line 193-203).

Let’s say trial(i) and trial(j) had different responses. We then will want to write the stimulus to the inconsistent. Of course we first have to check the stimulus is not already in the inconsistent stim array (stimsI). When adding the inconsistent stim we check if trial(j) had a response, if it did not we tag the stimuli with no response.

*Line 238-280: Getting consistent stims for test phase*

Similar to the study phase we start of by opening the files we need. This time though, we need the consistent stims (fidS) from the study phase. What we want to do is loop through each line of the fidS and until we find the trial with the matching stimulus.

Line 244 gets the first line of fidS and is stored in the variable line. Line 246 is a loop that continues as long as line has some string value. Now stimulus information in fidS is written with this structure: (stimulus name) (response type). So what we want to do in the loop, is for each value of line is split it at the tab. What we get then is a two element cell array (stimS) where stimS{1} is the stimulus name and stimS{2} is the response type. Ok so we want to iterate through all the trials (Line 250) until we find a trial in which trial(i).pseudoStim.stimID matches stimS{1} (Line 251). When we find a match the first thing we do is set the correct response key. If the participant consistently responded real to a pseudo stim (or vice versa) in the study phase we change the correct response to real (Line 253-259). After this we then check if the test phase response is consistent with the study phase responses (Line 262). If it is we add the trial to trialsAnalyzed and the stimulus stimulus to our array of consistent stimuli (stimsC) to be recorded into a file later (Line 263-267). If the response did not match we add the stimulus to the inconsistent stims array (stimsI) (Line 269-272).

If we are in practice trialAnalyzed will always be empty.

*Line 286-300: Recording stimuli to files*

Now that we have our arrays with consistent and inconsistent stimuli we record them to the appropriate files. I just used two for loops and iterated through stimsC and stimsI and wrote them to a file.

*Line 300-787: Behavioral Analysis*

Honestly this portion of the code is really self explanatory. So I will just a give a quick overview. Earlier I declared a variable for pretty much every stimulus scenario (ie RTPseudoStudy1 is the total reaction time of pseudo stimuli in study 1). So now I just want sort through the data to know to get the correct values I need for the RT analysis. There are 3 RT analyses we are doing: 1. All trials 2. Correct Trials 3. Consistent Trials. The first two is handled in the first for loop that iterates through every trial completed (Line 302). What happens in this loop is that it takes a trial and then through a series of if statements sorts what type of stimuli (real or pseudo) and what category (ie study 1 or study 2 in study phase) and includes it in the all the appropriate calculations (accuracy and RT for whatever groups it is in). It does the same for correct trials (which have their own set of variable ie correctRTRealOrtho). After sorting through the trials it does all the the calculation for accuracy and RT in every group you can imagine (line 558-670).

The third analysis is handled by looping through the trialsAnalyzed (line 674) which contains all the contains all the consistent stims. Inside the loop the code if fundamentally the same (some times I switched the order I checked things but all the same shit). You can probably see ways that this code could be condensed. Finally, we do the calculations in lines 735-785.