PyHab Documentation

Release v0.5

Jonathan F. Kominsky

Oct 04, 2022

Contents:

This is documentation for the code of PyHab. Eventually this will contain all of the documentation for PyHab, but that work is in progress.

CHAPTER 1

Installation

Prerequisites:: Install the most recent version of PsychoPy (https://www.psychopy.org/download.html).

You will also need VLC media player in order to play media. It is free. (https://www.videolan.org/vlc/index.html)

On Windows, you may need to install some movie codecs into PsychoPy directly. See WindowsTroubleshooting

Download the latest release of PyHab as a .zip file (https://github.com/jfkominsky/PyHab/releases).

Unzip the folder anywhere you would like. Then, see *Quickstart guide*

CHAPTER 2

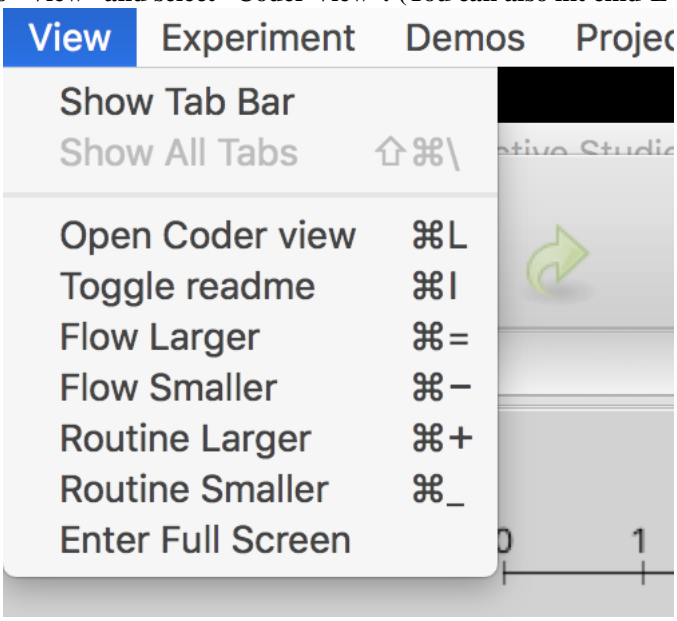
Quickstart guide

2.1 ManyBabies 4 setup

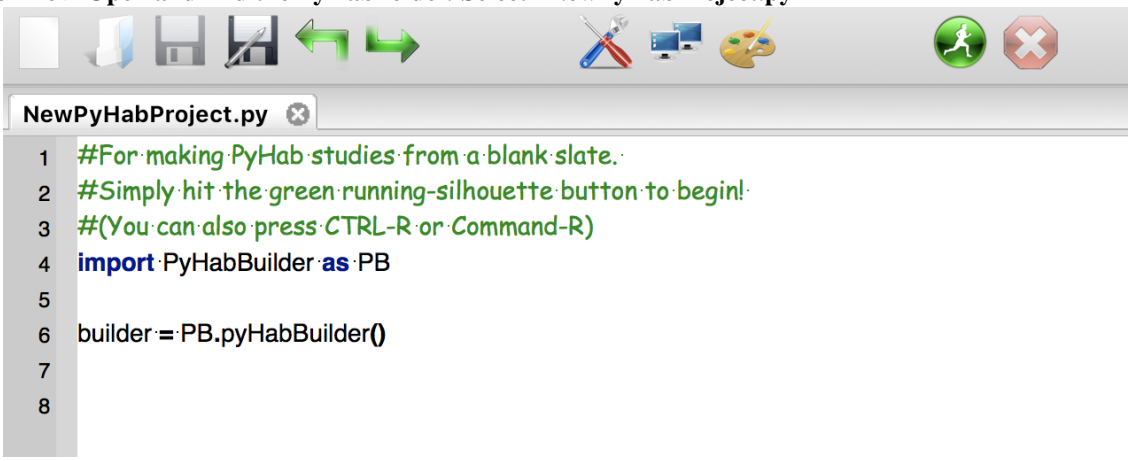
If you are helping out with the ManyBabies 4 project, there is a video guide to setting up the PyHab version of the ManyBabies 4 experiment, as well as a ManyBabies 4-specific text setup guide.

2.2 Creating a new PyHab project from scratch

1. Open PsychoPy
2. **Go to "View" and select "Coder View". (You can also hit cmd-L on Mac or ctrl-L on Windows.)**



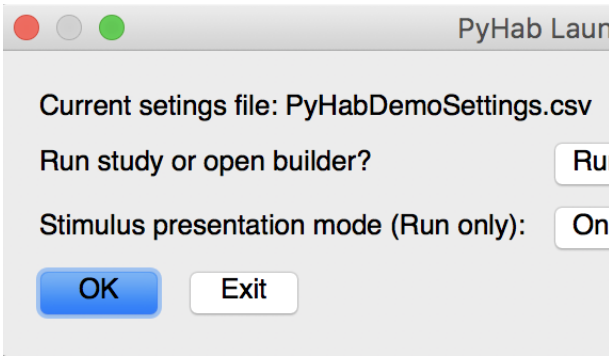
3. **Go to File > Open and find the PyHab folder. Select "NewPyHabProject.py"**



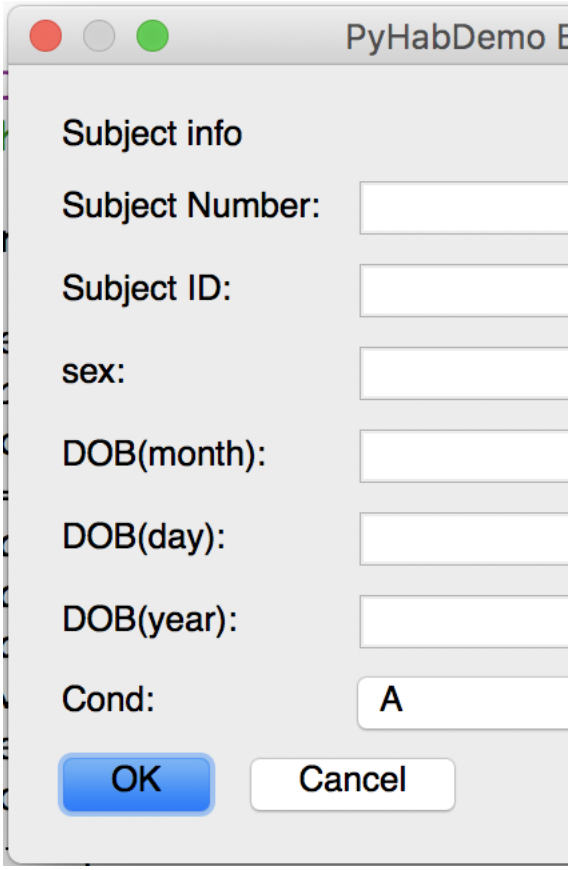
4. This should open a short text script in the coder window. Hit the big green "Run" button.
5. Construct your study and save it.
6. In the PsychoPy coder, choose "Open", then find the experiment folder you saved. Open the file "[project-name]Launcher.py"
7. Hit the big green run button to open the builder.
8. Read the manual (https://github.com/jfkominsky/PyHab/raw/master/PyHab%20User%20Manual.pdf) for more information on how to build an experiment.

2.3 Running the demo or a pre-made PyHab experiment

1. In coder view, open the experiment's launcher script. For the demo experiment, in the PyHab folder open "PyHabDemoLauncher.py" in the "PyHabDemo" folder.



2. **Hit the green "Run" button, and you should see a window like this.**
3. If you want to modify anything about the experiment, click the first drop-down menu and select "Builder", and then click OK. If you want to run the experiment, select whether or not you want it to present stimuli with the second drop-down list, and then hit OK.



4. **The next window that appears will be the subject information window.**

   4a. The first two lines are the subject number and identifier. These are arbitrary, but you need to put something in at least one of them.

   4b. The subject sex is entirely optional and can be left blank.

   4c. The next three lines are the subject date of birth. These MUST be filled out. Each field takes ONLY two digits. So, for a date of birth of November 12, 2018, you would put 11 in the first box, 12 in the second, and 18 in the third.

   4d. The final line is a condition drop-down menu with a list of conditions. If the experiment you are running does not have conditions configured, this will be an arbitrary text box. Otherwise, select the appopriate condition.

5. **Hit OK, and the experiment will launch. You will be presented with the experimenter window. Press "A" to start the first**
   5a. For preferential looking studies, hold "B" when they are looking left and "M" when they are looking right.

6. The experiment will notify you when it is saving data and then prompt you to press "return" to close the stimuli and experimenter windows and return to the launcher menu.

CHAPTER 3

Builder

The builder code uses PsychoPy's visual library to create a rudimentary GUI for creating PyHab studies. The GUI itself mostly consists of clickable shapes that open dialog boxes. The only notable exception is the system for creating conditions, which creates an entire new UI in the window.

When the builder's save functions are called, they create a complete, self-contained folder which includes the experiment setitngs file (a csv), a launcher script, the PyHab module folder with PyHabClass, PyHabClassPL, and PyHabBuilder, and copies of all of the stimuli and attention-getters to a stimuli folder.

class PyHab.PyHabBuilder.**PyHabBuilder**(*loadedSaved=False*, *settingsDict={}*)

Graphical interface for constructing PyHab experiments. Runs mostly on a Pyglet window and qtGUI dialogs. Saves a settings file in .csv form which can then be read by PyHab Launcher, PyHabClass, PyHabClassPL, and PyHabClassHPP.

HPP_stimSettingsDlg()

A dialog box for the stimulus settings unique to head-turn preference procedures. Just selects the screen to then open the stimulus dialog window for.

Returns

Return type

addStimToLibraryDlg()

A series of dialog boxes which allows you to build a "library" of stimulus files for your experiment, which you can then assign to trial types in a separate dialog.

Works a bit like the attention-getter construction dialogs, but different in that it allows audio or images alone. The image/audio pairs are complicated, but not worth splitting into their own function at this time.

Returns

Return type

addStimToTypesDlg()

A series dialog boxes, the first selecting a trial type and the number of stimuli to add to it, a second allowing you to add stimuli from the stimulus library that is stimList in the settings. Also used for adding beginning and end of experiment images

Returns

Return type

advTrialDlg(*trialType*)

A dialog for advanced trial settings mostly having to do with attention-getters and stimulus presentation

0/-8 = cutoff attention-getter on gaze-on? T/F [if trial has an AG - not always present]

1/-7 = cutoff on-time: How long do they have to look to cut off presentation? [if trial has an AG - not always present]

2/-6 = Pause stimulus presentation when infant is not looking at screen?

3/-5 = Mid-trial AG: Play an attention-getter if infant looks away mid-trial?

4/-4 = mid-trial AG trigger: Minimum time to trigger mid-trial AG

5/-3 = mid-trial AG cutoff: Stop mid-trial AG on gaze-on?

6/-2 = mid-trial AG cutoff ontime

7/-1 = HPP ONLY: Only count gaze-on to stimulus-presenting screen? T/F, casts to hppStimScrOnly

Parameters **trialType** (*str*) – The trial type being modified.

Returns

Return type

advTrialSetup()

A function for selecting the trials you want to access the advanced settings of. Spawns a panel with all the trial types. Reuses code from the block interface. Doesn't need to check existence of trials because

Returns

Return type

attnGetterAudioDlg()

A modular dialog for setting the options for an audio-based attention-getter

Returns A dictionary containing all the info required for an audio attention-getter

Return type dict

attnGetterDlg()

The dialog window for customizing the attention-getters available to use for different trials. Two-stage: Modify existing attngetter or make new, then what do you do with ether of those. Allows audio with PsychoPy-produced looming shape or just a video file.

Returns

Return type

attnGetterMovieAudioDlg()

A modular dialog for finding an audio file and a video file and titling them appropriately.

Returns A dictionary containing all the info required for a video/audio attngetter.

Return type

attnGetterVideoDlg()

A modular dialog for setting the options for a video-based attention-getter

Returns A dictionary containing all the info required for a video attention-getter

Return type dict

autoCond(*genTrials*, *genBlocks*, *counters*)

The function that actually creates conditions automatically, given what it is generating conditions for, and how many items from each trial/block it should have in each condition. Simply sets condDict and condList.

Parameters

- **genTrials** (*bool*) – Are we generating conditions for movies within trial types?
- **genBlocks** (*bool*) – Are we generating conditions for trials within blocks?
- **counters** (*dict*) – A dictionary of each block or trial type that needs to be randomized and the number of items that thing should have in each condition

Returns

Return type

autoCondSetup()

Function for getting the parameters for automatically generating conditions. A series of dialogs. The first one, whatGenDlg, determines whether we are doing stim within trials, trials within blocks, or both. It also determines whether we are keeping all items, which shortcuts the second dialog. whatGenDlg: 0 (if blocks): Randomize order of stimuli in trials y/n 1 (if blocks): Randomize order of trials in blocks y/n -1: Keep all items in all conditions y/n

A second dialog then asks whether there are any trial/block types the user does not want to randomize.

If not keeping all items in all conditions, another dialog is needed to determine size of subset for each trial/block. This produces a dictionary that tracks how many items will be in each condition. :return: :rtype:

blockDataDlg()

A dialog for determining whether you save a block data file, and if so which blocks to compress.

Procedurally constructs a set of options such that, for any nested blocks, they are mutually exclusive, but any blocks that are not part of other blocks and other blocks are not part of them are just check-boxes.

Excludes hab because habituation data files are saved by default.

Returns

Return type

blockMaker(*blockName*, *new=True*, *hab=False*)

For making multi-trial blocks. Or multi-block-blocks. Blocks are necessary for habituation.

Creates a kind of sub-UI that overlays over the main UI. Because it's just for blocks, we can ditch some things. We can actually completely overlay the regular UI. Problem is, if the regular UI continues to draw, the mouse detection will still work, even if a shape is behind another shape. So, like with conditions, we need a totally parallel UI

Hab blocks cannot be embedded in other blocks.

Parameters

- **blockName** (*str*) – Name of new block
- **new** (*bool*) – Is this a new block or a modification of an existing one?
- **hab** (*bool*) – Is this for a habituation meta-trial? # TODO: No longer needs to be an argument?

Returns

Return type

condMaker(*rep=False*, *currPage=1*, *bc=False*, *trialMode=True*, *resetDict={}*)
A whole separate interface for managing condition creation.

Outputs settings condList (labels of each condition), condFile (save conditions to this file) and makes new structure condDict (mapping of each label to actual condition it applies to)

Parameters

- **rep** (*bool*) – Basically whether we are recursing while editing conditions
- **currPage** (*int*) – The current page number
- **bc** (*bool*) – Are we displaying the raw conditions, or the 'base' (pre-randomization) conditions?
- **trialMode** (*bool*) – A toggle between 'trial mode' (use conditions for order of stimuli in trial types) and 'block mode' (use conditions to change order of trials within blocks)

Returns

Return type

condRandomizer()
This is based on other scripts I've made. Basically, say you have four conditions, and you want four participants to be assigned to each one, but you want to be totally blind to which condition a given participant is in. Here, once you have made your four conditions, you can tell it to create a condition list that it never shows you that has each condition X times, and that becomes the new condition file/list/etc.

Returns

Return type

condSetter(*shuffleList*, *cond='NEW'*, *ex=False*, *blockmode=False*)
A new interface for ordering stimuli within a trial type or trials within a block for a specific condition. Increases flexibility and usability. Uses an overlay like the block-constructor interface

ST/PL output: {trialType:[stim1, stim2]} HPP output: {trialType:[{'L':0,'C':stim1,'R':0},{}]}

Parameters

- **shuffleList** (*dict*) – Either the stimNames dict or the blockList dict. Defines which one we are modifying.
- **cond** (*str*) – Condition name
- **ex** (*bool*) – Whether the condition already exists
- **blockmode** (*bool*) – Are we reordering a block or a trial? Matters because even in HPP need to be blocks

Returns

Return type

condSettingsDlg()
The dialog window for "condition settings", not to be confused with the condition interface created by self.condMaker(). This determines whether condition randomization is used at all, a separate interface is used to define the conditions themselves.

Returns

Return type

dataSettingsDlg()
Which columns of data are recorded. Resets if the experiment type is switched to or from preferential looking.

Returns

Return type

delCond()
Present list of existing conditions. Choose one to remove.

delTrialTypeDlg()
Dialog for deleting a trial type, and all instances of that trial type in the study flow

Returns

Return type

deleteType(*dType*)
Performs the actual deletion of a trial or block type. TODO: More sophisticated handling of conditions.

Parameters **dType** (*str*) – String indicating the name of the trial or block to be deleted

Returns

Return type

habSettingsDlg(*trialList*, *lastSet*, *redo=False*)
Dialog for settings relating to habituation criteria:

0 = habituation (Active/not active)

1 = maxHabTrials (maximum possible hab trials if criterion not met)

2 = setCritWindow (# trials summed over when creating criterion)

3 = setCritDivisor (denominator of criterion calculation . e.g., sum of first 3 trials divided by 2 would have 3 for setCritWindow and 2 for this.)

4 = setCritType (peak window, max trials, first N, last N, or first N above threshold)

5 = habThresh (threshold for N above threshold)

6 = metCritWindow (# trials summed over when evaluating whether criterion has been met)

7 = metCritDivisor (denominator of sum calculated when determining if criterion has been met)

8 = metCritStatic (static or moving window?)

9 = habByDuration (habituation by duration or by on-time)

10-N = Which trials to calculate hab over for multi-trial blocks.

Parameters

- **trialList** (*list*) – List of available trials in the block, since this follows from block settings.
- **lastSet** (*dict*) – If information entered is invalid and the dialog needs to be shown again, this allows it to remember what was previously entered. Also pulls from existing block settings.
- **redo** (*boolean*) – Checking if redoing last setting

Returns A dictionary of settings fed back into the block-maker UI.

Return type dict

lastPalettePage()
Simple function for moving to the previous page of the trial type palette :return: :rtype:

loadFlow(*tOrd*, *space*, *locs*, *overflow*, *types*, *conlines=True*, *trials=True*, *specNumItems=0*)
Creates the array of objects to be drawn for a study flow or block flow.

Flow dictionary components: 'lines': Lines that go between items in the flow, drawn first 'shapes': visual.Rect objects 'text': visual.textStim objects 'labels': Strings that label each trial. Shapes and text are indexted to these, so you can do easy lookup. 'extras': Special category for trial pips for blocks.

Parameters

- **tOrd** (*list*) – Extant order of trials, either the overall trial order or the block order
- **space** (*list*) – The dimensions of the flow part of the UI, typically self.flowArea
- **locs** (*list*) – List of locations to draw the items in the flow, if less than 21 items to be drawn
- **overflow** (*list*) – List of locations to use when there are more than 21 items, which compacts the rendering.
- **types** (*list*) – List of items being put into this flow. Typically the list of trial types, except when making conditions.
- **conlines** – A special boolean added for cases where we might not want connecting lines between items in the flow, e.g. conditions.
- **trials** (*bool*) – A special bool added for cases where we might not be dealing with trial types, to avoid certain pitfalls in conditional logic.
- **specNumItems** (*int*) – A special argument for cases where there are weird line overlaps that change the length of things. Defaults to 0, only used when calling recursively.

Returns A dictionary of all of the entities to draw into the block or study flow

Return type dict

loadTypes(*typeLocations*, *typeList*, *page=1*)
This function creates the trial types palette.

Type pallette dictionary components: 'shapes': visual.Rect objects 'text': visual.TextStim objects 'labels': A sort of index for the other two, a plain string labeling the trial or block type.

Parameters

- **typeLocations** (*list*) – The array of coordinates on which buttons can be placed. Usually self.typeLocs
- **typeList** (*list*) – A list of what is being populated into the array. Usually self.settings['trialTypes']
- **page** (*int*) – Page number for when there are more types than fit on one page, in order to render the correct one.

Returns

Return type

mainLoop()
Main loop of the whole program.

Returns

Return type

makeBlockDlg(*name=''*, *new=True*)
Creates a new 'block' structure, which basically masquerades as a trial type in most regards, but consists of several sub-trials, much like how habituation blocks work.

Parameters

- **name** (*str*) – Name of existing trial type. '' by default
- **new** (*bool*) – Making a new block, or modifying an existing one?

Returns

Return type

moveTrialInFlow(*flowIndex*, *tOrd*, *flowSpace*, *UI*, *flow*, *types*)

A function for when a trial is clicked in a trial flow, allowing you to either swap it or remove it.

Parameters

- **flowIndex** (*int*) – The index in the flowArray of the trial being modified
- **tOrd** (*list*) – The trial order being modified, either the main one or a block order
- **flowSpace** (*visual.Rect object*) – The shape that makes up the flow UI, which varies from typical usage to block construction
- **UI** (*dict*) – A dictionary containing the currently active UI
- **flow** (*dict*) – A dictionary containing the currently active trial flow
- **types** (*dict*) – A dictionary containing the currently active trial pallette (mostly for showMainUI)

Returns The modified trial order

Return type list

nextPalettePage()

Simple function for moving to the next page of the trial type palette. :return: :rtype:

quitFunc()

Simple function for quitting, checks if you want to save first (if there's anything to save).

Returns

Return type

removeStimFromLibrary()

Presents a dialog listing every item of stimuli in the study library. Allows you to remove any number at once, removes from all trial types at same time. Deletes from stimuli folder on save if extant.

Returns

Return type

run()

Exists exclusively to be called to start the main loop.

Returns

Return type

saveDlg()

Opens a save dialog allowing you to choose where to save your project. Essentially sets self.folderPath

Returns

Return type

saveEverything()

Saves a PyHab project to a set of folders dictated by self.folderPath

todo: Add psychopy_tobii_infant to this. Saved in the code folder.

Returns

Return type

showMainUI(*UI*, *flow*, *types*)

A simple function that draws everything and flips the display. Generalized to work for block mode and general mode.

Parameters **UI** – a dictionary of everything to be drawn in the new UI. Contains a list, 'bg' (background), and a dict,

'buttons', that has itself 'shapes' and 'text' (and usually 'functions' but this doesn't need to know that) :type UI: dict :param flow: A dict of everything in the block flow. Contains five lists, 'lines', 'shapes', 'text', 'labels', and 'extra' :type flow: dict :param types: A dict of the trial type buttons for this block. Contains three lists: 'shapes', 'text', and 'labels' :type types: dict :return: :rtype:

stimSettingsDlg(*lastSet=[]*, *redo=False*, *screen='all'*)

Settings relating to stimulus presentation. Indexes from the dialog (non-HPP version):

0 = screenWidth: Width of stim window

1 = screenHeight: Height of stim window

2 = Background color of stim window

3 = movieWidth: Width of movieStim3 object inside stim window. Future: Allows for movie default resolution?

4 = movieWidth: Height of movieStim3 object inside stim window

5 = freezeFrame: If the attention-getter is used (for a given trial type), this is the minimum time the first frame of the movie will be displayed after the attention-getter finishes.

6 = screenIndex: Which screen to display the stim window on.

7 = expScreenIndex: Which screen to display the experimenter window on

Parameters

- **lastSet** (*list*) – Optional. Last entered settings, in case dialog needs to be presented again to fix bad entries.
- **redo** (*boolean*) – Are we doing this again to fix bad entries?
- **screen** (*string*) – Optional. For HPP, lets you set for just the individual screen the settings apply to.

Returns

Return type

toHPP()

A function that converts ST or PL experiments to HPP. Always a little complicated.

Returns

Return type

toPL()

A function that converts ST or HPP to preferential looking. ST to PL is NBD. HPP is more of a challenge. :return: :rtype:

toST()

A function that converts PL or HPP to single-target. PL to ST is NBD. HPP is more of a challenge.

Returns

Return type

trialTypeDlg(*trialType='TrialTypeNew'*, *makeNew=True*, *prevInfo=[]*)

Dialog for creating OR modifying a trial type. Allows you to set the maximum duration of that trial type as well as remove movies from it, and also set whether the trial type is gaze contingent. Now also sets whether the study should auto-advance into this trial and whether the built-in attention-getter should be used.

The dialog by default outputs a list with 12 items in it. 0 = trial type name

1 = Maximum duration of trials of this type

2 = Gaze-contingent trial type?

3 = Maximum continuous looking-away to end trial of type

4 = Minimum on-time (cumulative)

5 = auto-redo trial if minimum on-time not met?

6 = on-time deadline

7 = duration criterion rather than on-time

8 = Auto-advance into trial?

9 = Attention-getter selection

10 = End trial on movie end or mid-movie

11 = inter-stimulus interveral (ISI) for this trial type

12 = Maximum on-time (single use case, for new gaze contingent trial type mode).

[if movies assigned to trial type already, they occupy 13 - N]

Parameters

- **trialType** (*str*) – Name of the trial type
- **makeNew** (*bool*) – Making a new trial type or modifying an existing one?
- **prevInfo** (*list*) – If user attempts to create an invalid trial type, the dialog is re-opened with the previously entered information stored and restored

Returns

Return type

univSettingsDlg()

Settings that apply to every PyHab study regardless of anything else.

0 = prefix: The prefix of the launcher and all data files.

1 = blindPres: Level of experimenter blinding, 0 (none), 1 (no trial type info), or 2 (only info is whether a trial is currently active.

2 = nextFlash: Whether to have the coder window flash to alert the experimenter they need to manually trigger the next trial

3 = durationInclude: Trial duration calculations include last gaze-off or not

4 = loadSeparate: New setting in 0.9.4, movie playback issues have created a situation where some (but not all) experiments might benefit from going back to the old ways of loading one movie file for *each* individual instance of a trial, rather than trying to load one movie file and load it once. This setting controls whether that happens.

5 = eyetracker: New in 0.10.4, Tobii integration (which is much more seamless than alternatives). Can be set to simply record eye-tracking info OR to control the experiment as a replacement for a human coder. (0/1/2)

Returns

Return type

CHAPTER 4

PyHab Class (base)

This is the base class that is used to actually run PyHab studies. There is an extension of this base class for preferential looking paradigms, see *Preferential Looking*

class PyHab.PyHabClass.**PyHab**(*settingsDict*, *testMode=False*)

PyHab looking time coding + stimulus control system

Jonathan Kominsky, 2016-2018

Keyboard coding: A = ready, B = coder 1 on, L = coder 2 on, R = abort trial, Y = end experiment (for fussouts)

Between-trials: R = redo previous trial, J = jump to test trial, I = insert additional habituation trial (hab only)

Throughout this script, win2 is the coder display, win is the stimulus presentation window. dataMatrix is the primary data storage for the summary data file. It is a list of dicts, each dict corresponds to a trial.

Anything called "verbose" is part of the verbose data file. There are up to four such structures: On (for gaze-on events) Off (for gaze-off events) On2 and Off2 (for the optional secondary coder) Each coder's on and off are recorded in a separate dict with trial, gaze on/off, start, end, and duration.

SetupWindow()

Sets up the stimulus presentation and coder windows, loads all the stimuli, then starts the experiment with doExperiment()

Returns

Return type

TrackerCalibrateValidate()

Function that controls the eye-tracker calibration and validation in eye-tracking modes. In principle this can be run mid-experiment, but it will be disruptive

Returns

Return type

abortTrial(*onArray*, *offArray*, *trial*, *ttype*, *onArray2*, *offArray2*, *stimName=''*, *habTrialNo=0*, *habCrit=0.0*)

Only happens when the 'abort' button is pressed during a trial. Creates a "bad trial" entry out of any data recorded for the trial so far, to be saved later.

Parameters

- **onArray** (*list of dicts {trial, trialType, startTime, endTime, duration}*) – Gaze-on events for coder 1
- **offArray** (*list of dicts {trial, trialType, startTime, endTime, duration}*) – Gaze-off events for coder 1
- **trial** (*int*) – trial number
- **ttype** (*string*) – trial type
- **onArray2** (*list of dicts*) – Gaze-on events for (optional) coder 2
- **offArray2** (*list of dicts*) – Gaze-off events for (optional) coder 2
- **stimName** (*string*) – If presenting stimuli, name of the stim file
- **habTrialNo** (*int*) – Tracking if this is a habituation trial and if so what number
- **habCrit** (*float*) – Habituation criterion, if it's been set

Returns

Return type

attnGetter(*trialType*, *cutoff=False*, *onmin=0.0*, *midTrial=False*)

Plays either a default attention-getter animation or a user-defined one. Separate settings for audio w/shape and video file attention-getters.

Parameters

- **trialType** (*string*) – Current trial type
- **cutoff** (*bool*) – Cut off AG immediately on gaze-on? Defaut False
- **onmin** (*float*) – Delay in listening for gaze-on to immediately end AG. Default 0
- **midTrial** (*bool*) – Is this a mid-trial attention-getter? Default False

Returns

Return type

avgObsAgree(*timewarp*, *timewarp2*)

Computes average observer agreement as agreement in each trial, divided by number of trials.

Parameters

- **timewarp** (*list*) – List of every individual frame's gaze-on/gaze-off code for coder A
- **timewarp2** (*list*) – As above for coder B

Returns average observer agreement or N/A if no valid data

Return type float

blockExpander(*blockInfo*, *prefixes*, *hab=False*, *habNum=0*, *insert=-1*, *baseStart=-1*)

A method for constructing actualTrialOrder while dealing with recursive blocks. Can create incredibly long trial codes, but ensures that all information is accurately preserved. Works for both hab blocks and other things.

For hab blocks, we can take advantage of the fact that hab cannot appear inside any other block. It will always be the top-level block, and so we can adjust the prefix once and it will carry through.

The trial naming preserves hierarchy in a block.trial or block.subblock.trial form. Hab blocks are designated by a '*' before the first '.', and the last trial in a hab block is marked with '^', which is needed to trip checkStop.

Because hab blocks cannot be embedded in other blocks, the top-level block is always the one with the hab settings.

Parameters

- **blockInfo** (*dict*) – The data of the block object, including trialList and hab info.
- **prefixes** (*str*) – A recursively growing stack of prefixes. If block A has B and block B has C, then an instance of A will be A.B.C in self.actualTrialOrder. This keeps track of the A.B. part.
- **hab** (*bool*) – Are we dealing with a habituation trial expansion?
- **habNum** (*int*) – If we are dealing with a habituation trial expansion, what hab iteration of it are we on?
- **insert** (*int*) – An int specifying where in actualTrialOrder to put a trial. Needed to generalize this function for insertHab
- **baseStart** (*int*) – Marks the index where the top-level block started in actualTrialOrder.

Returns

Return type

checkStop(*blockName*)

After a hab trial, checks the habitution criteria and returns 'true' if any of them are met. Also responsible for setting the habituation criteria according to settings. Prior to any criteria being set, self.HabCrit is 0, and self.habSetWhen is -1.

Uses a sort of parallel data structure that just tracks hab-relevant gaze totals. As a bonus, this means it now works for both single-target and preferential looking designs (and HPP designs) with no modification.

To support multiple hab blocks, this needs to take the block name as an argument, to only look at that block's hab settings. That means each block with habituation turned on can only be used once, but you can have more than one block

Parameters **blockName** (*str*) – The name of the block associated with the hab trial, required to look up its particular settings.

Returns True if hab criteria have been met, False otherwise

Return type bool

cohensKappa(*timewarp*, *timewarp2*)

Computes Cohen's Kappa

Parameters

- **timewarp** (*list*) – List of every individual frame's gaze-on/gaze-off code for coder A
- **timewarp2** (*list*) – As above for coder B

Returns Kappa

Return type float

dataRec(*onArray*, *offArray*, *trial*, *type*, *onArray2*, *offArray2*, *stimName=''*, *habTrialNo=0*, *habCrit=0.0*)

Records the data for a trial that ended normally.

Parameters

- **onArray** (*list of dicts {trial, trialType, startTime, endTime, duration}*) – Gaze-on events for coder 1

- **offArray** (*list of dicts {trial, trialType, startTime, endTime, duration}*) – Gaze-off events for coder 1
- **trial** (*int*) – trial number
- **type** (*string*) – trial type
- **onArray2** (*list*) – Gaze-on events for (optional) coder 2
- **offArray2** (*list*) – Gaze-off events for (optional) coder 2
- **stimName** (*string*) – If presenting stimuli, name of the stim file
- **habTrialNo** (*int*) – If part of a hab block, what hab trial it was part of.
- **habCrit** (*double*) – If part of a hab block, the current habituation criterion.

Returns

Return type

dispAudioStim(*trialType*, *dispAudio*)

For playing audio stimuli. A little more complicated than most because it needs to track whether the audio is playing or not. Audio plays separately from main thread.

Parameters **dispAudio** (*sound.Sound object*) – the stimuli as a sound.Sound object

Returns an int specifying whether the audio is in progress (0), we are in an ISI (1), or the audio is looping (2)

Return type int

dispCoderWindow(*trialType=-1*)

Draws the coder window, according to trial type and blinding settings.

Parameters **trialType** (*int or string*) – -1 = black (betwen trials). 0 = ready state. Otherwise irrelevant.

Returns

Return type

dispImageStim(*dispImage*, *screen='C'*)

Very simple. Draws still-image stimuli and flips window

Parameters

- **dispImage** (*visual.ImageStim object*) – the visual.ImageStim object
- **screen** (*str*) – For HPP, which screen the image is to appear on

Returns constant, 1

Return type int

dispMovieStim(*trialType*, *dispMovie*, *screen='C'*)

Draws movie stimuli to the stimulus display, including movie-based attention-getters.

Parameters

- **trialType** (*int or str*) – 0 for paused, otherwise a string
- **dispMovie** (*moviestim3 object*) – The moviestim3 object for the stimuli
- **screen** (*str*) – The screen on which the movie should display. Only relevant for HPP.

Returns an int specifying whether the movie is in progress (0), paused on its last frame (1), or ending and looping (2)

Return type int

dispTrial (*trialType*, *dispMovie=False*)

Draws each frame of the trial. For stimPres, returns a movie-status value for determining when the movie has ended

Parameters

- **trialType** (*string*) – Current trial type
- **dispMovie** (*bool or dict*) – A dictionary containing both the stimulus type and the object with the stimulus file(s) (if applicable)

Returns 1 or 0. 1 = end of movie for trials that end on that.

Return type int

doExperiment ()

The primary control function and main trial loop.

Returns

Return type

doTrial (*number*, *ttype*, *disMovie*)

Control function for individual trials, to be called by doExperiment Returns a status value (int) that tells doExperiment what to do next

self.playThrough registers the end-trial crieria 0 = standard "cumulative on-time >= MinOn and consecutive off-time >= MaxOff" 1 = "OnOnly", only requires that cumulative on-time > MinOn 2 = "None", plays to max duration no matter what. 3 = "Either/or", as standard but with "or" instead of "and". Whichever comes first.

Parameters

- **number** (*int*) – Trial number
- **ttype** (*string*) – Trial type - the full expanded one with block hierarchy and hab trial info included.
- **disMovie** (*dictionary*) – A dictionary as follows { 'stim':[psychopy object for stimulus presentation], 'stimType':[movie,image,audio, pair]}

Returns int, 0 = proceed to next trial, 1 = hab crit met, 2 = end experiment, 3 = trial aborted

Return type

endExperiment ()

End experiment, save all data, calculate reliability if needed, close up shop. Displays "saving data" and end-of-experiment screen.

Returns

Return type

flashCoderWindow (*rep=False*)

Flash the background of the coder window to alert the experimenter they need to initiate the next trial. .2 seconds of white and black, flashed twice. Can lengthen gap between trial but listens for 'A' on every flip.

Returns

Return type

insertHab (*tn*, *block*, *hn=-1*)

Literally insert a new hab trial or meta-trial into actualTrialOrder, get the right movie, etc.

Parameters

- **tn** (*int*) – trial number to insert the trial
- **hn** – HabCount number to insert the hab trial. By default, whatever the current habcount is. However, there

are edge cases when recovering from "redo" trials when we want to throw in a hab trial further down the line. :type hn: int :param block: The habituation block the trial is being added to :type block: str :return: [disMovie, trialType], the former being the movie file to play if relevant, and the latter being the new trial type :rtype: list

isInt()

silly little function for validating a very narrow usage of "cond" field

Returns Bool: if arbitrary argument t is an int, true.

Return type

jumpToTest(*tn*, *block*, *met=False*)

Jumps out of a hab block into whatever the first trial after the current hab block.

Parameters

- **tn** (*int*) – current trial number when the function is called
- **block** (*str*) – Block the habituation belongs to. Find end of block, done, because hab blocks can't be embedded and each one can only occur in the flow once.
- **met** (*bool*) – A boolean for whether this is because of J (False) or whether this is a genuine hab-criterion-met

Returns [disMovie, trialType] as insertHab, the former being the movie file to play if relevant, and the latter being the new trial type

Return type list

loadStim(*stim*, *screen='C'*)

A general function for loading stimuli that can be called repeatedly.

TODO: Windows audio bug when loading an audio file before a movie file means that we should change load order for everything to movie first.

Parameters

- **stim** (*str*) – stimulus name, key for stimList dict
- **screen** (*str*) – Screen to load stimuli to, doesn't matter except for HPP, defaults to center

Returns a dictionary with type and stimulus object

Return type dict

lookKeysPressed()

A simple boolean function to allow for more modularity with preferential looking Basically, allows you to set an arbitrary set of keys to start a trial once the attngetter has played. In this case, only B (coder A on) is sufficient.

This function can become the eye-tracker interface, basically. It will listen for the eye-tracker input.

Todo: Can we implement a debug mode that simulates key-presses for some amount of time?

Returns True if the B key is pressed, False otherwise.

Return type bool

lookScreenKeyPressed(*screen=['C']*)
A function that primarily exists for HPP, because of the need to distinguish between any key being pressed and the key corresponding to the HPP screen in question being pressed

Parameters **screen** (*list of strings*) – List of screens for next stim.

Returns for non-HPP versions, the value of lookKeysPressed.

Return type bool

pearsonR(*verboseMatrix*, *verboseMatrix2*)
Computes Pearson's R

Parameters

- **verboseMatrix** (*dict*) – Verbose data, coder A
- **verboseMatrix2** (*dict*) – Verboase data, coder B

Returns Pearson's R

Return type float

printCurrentData()
A function which prints the current data to the output window, made into its own function to facilitate having working versions for PL and HPP studies as well. Only called when stimulus presentation is off.

Returns

Return type

redoSetup(*tn*, *autoAdv*, *blockName*, *blockRedo=False*, *fromAbort=False*)
Lays the groundwork for redoTrial, including correcting the trial order, selecting the right stim, etc.

Parameters

- **tn** (*int*) – Trial number (trialNum), to be redone (or in the process of being aborted)
- **autoAdv** (*list*) – The current auto-advance trial type list (different on first trial for Reasons)
- **blockName** (*str*) – Pulls topBlockName from doExperiment to deal with redoing block and habituations.
- **blockRedo** (*bool*) – A special type of redo reserved for blocks, that rewinds to the start of the top-level block.
- **fromAbort** (*bool*) – A bool for blockRedos, to see if this is coming off a redo, which requires an extra check.

Returns list, [disMovie, trialNum], the former being the movie file to play if relevant, and the latter being the new trial number

Return type

redoTrial(*trialNum*)
Allows you to redo a trial after it has ended. Similar to abort trial, but under the assumption that the data has already been recorded and needs to be replaced.

This function only handles the data part. The actual re-assignment of the trial is done elsewhere.

Parameters **trialNum** (*int*) – Trial number to redo

Returns

Return type

reliability(*verboseMatrix*, *verboseMatrix2*)

Calculates reliability statistics. Constructed originally by Florin Gheorgiu for PyHab, modified by Jonathan Kominsky.

Parameters

- **verboseMatrix** (*list*) – A 2-dimensional list with the content of the verbose data file for coder 1
- **verboseMatrix2** (*list*) – A 2-dimensional list with the content of the verbose data file for coder 2

Returns A dict of four stats in float form (weighted % agreement, average observer agreement, Cohen's Kappa, and Pearson's R)

Return type dict

run(*testMode=[]*)

Startup function. Presents subject information dialog to researcher, reads and follows settings and condition files. Now with a testing mode to allow us to skip the dialog and ensure the actualTrialOrder structure is being put together properly in unit testing.

Also expands habituation blocks appropriately and tags trials with habituation iteration number as well as the symbol for the end of a hab block (^)

Parameters **testMode** (*list*) – Optional and primarily only used for unit testing. Will not launch the window and start the experiment. Contains all the info that would appear in the subject info dialog.

Returns

Return type

saveBlockFile()

A function that create a block-level summary file and saves it. Copies the primary data matrix (only good trials) and loops over it, compressing all blocks. Does not work for habs, which follow their own rules.

Returns A condensed copy of dataMatrix with all blocks of the relevant types condensed to one line.

Return type list

saveHabFile()

Creates a habituation summary data file, which has one line per hab trial, and only looks at parts of the hab trial that were included in calcHabOver. This is notably easier in some ways because the hab trials are already tagged in dataMatrix

Returns A condensed copy of dataMatrix with all hab trials condensed only to those that were used to compute habituation.

Return type list

wPA(*timewarp*, *timewarp2*)

Calculates weighted percentage agreement, computed as number of agreement frames over total frames.

Parameters

- **timewarp** (*list*) – List of every individual frame's gaze-on/gaze-off code for coder A
- **timewarp2** (*list*) – As above for coder B

Returns Weighted Percentage Agreement

Return type float

CHAPTER 5

Preferential Looking

This class is an extension of the *PyHab Class (base)* base class.

class PyHab.PyHabClassPL.**PyHabPL**(*settingsDict*, *testMode=False*)

A new preferential-looking version of PyHab that extends the base class rather than being a wholly separate class. There's still a lot of redundant code here, which will require significant restructuring of the base class to fix.

abortTrial(*onArray*, *offArray*, *trial*, *ttype*, *onArray2*, *stimName=''*, *habTrialNo=0*, *habCrit=0.0*)

Aborts a trial in progress, saves any data recorded thus far to the bad-data structures

Parameters

- **onArray** (*list of dicts {trial, trialType, startTime, endTime, duration}*) – Gaze-on Left events
- **offArray** (*list of dicts {trial, trialType, startTime, endTime, duration}*) – Gaze-off events
- **trial** (*int*) – Trial number
- **ttype** (*string*) – Trial type
- **onArray2** (*list of dicts {trial, trialType, startTime, endTime, duration}*) – Gaze-on Right events
- **stimName** (*string*) – If presenting stimuli, name of the stim file
- **habTrialNo** (*int*) – Tracking if this is a habituation trial and if so what number
- **habCrit** (*float*) – Habituation criterion, if it's been set

Returns

Return type

dataRec(*onArray*, *offArray*, *trial*, *type*, *onArray2*, *stimName=''*, *habTrialNo=0*, *habCrit=0.0*)

Records the data for a trial that ended normally.

Parameters

- **onArray** (*list of dicts {trial, trialType, startTime, endTime, duration}*) – Gaze-on Left events
- **offArray** (*list of dicts {trial, trialType, startTime, endTime, duration}*) – Gaze-off events
- **trial** (*int*) – Trial number
- **ttype** (*string*) – Trial type
- **onArray2** (*list of dicts {trial, trialType, startTime, endTime, duration}*) – Gaze-on Right events
- **stimName** (*string*) – If presenting stimuli, name of the stim file
- **habTrialNo** (*int*) – Tracking if this is a habituation trial and if so what number
- **habCrit** (*float*) – Habituation criterion, if it's been set

Returns

Return type

doTrial(*number*, *ttype*, *disMovie*)

Control function for individual trials, to be called by doExperiment Returns a status value (int) that tells doExperiment what to do next

Parameters

- **number** (*int*) – Trial number
- **ttype** (*string*) – Trial type
- **disMovie** (*dictionary*) – A dictionary as follows {'stim':[psychopy object for stimulus presentation], 'stimType':[movie,image,audio, pair]}

Returns int, 0 = proceed to next trial, 1 = hab crit met, 2 = end experiment, 3 = trial aborted

Return type

endExperiment()

End experiment, save all data, calculate reliability if needed, close up shop :return: :rtype:

lookKeysPressed()

A simple boolean function to allow for more modularity with preferential looking Basically, allows you to set an arbitrary set of keys to start a trial once the attngetter has played. In this case, only B or M are sufficient.

Returns True if the B or M key is pressed, False otherwise.

Return type

printCurrentData()

Prints the current data, preferential looking variant. Only called when stimulus presentation is off :return: :rtype:

CHAPTER 6

Head-turn Preference Procedure

This class is an extension of the *PyHab Class (base)* base class.

class PyHab.PyHabClassHPP.**PyHabHPP**(*settingsDict*, *testMode=False*)

A head-turn preference procedure version of PyHab. Uses some of the same code, but drastically more complex, needing to juggle which screen things are presented on, simultaneous presentation on multiple screens, and more.

SetupWindow()

An HPP-specific version of the function that sets up the windows and loads everything. With four windows to set up it's a real doozy, and has the added problem of needing to assign things properly to each window for stim presentation.

TODO: Windows audio bug when loading an audio file before a movie means that we should change load order to movie first

Returns

Return type

abortTrial(*onArray*, *offArray*, *trial*, *ttype*, *onArrayL*, *onArrayR*, *stimName=''*, *habTrialNo=0*, *habCrit=0.0*)

Aborts a trial in progress, saves any data recorded thus far to the bad-data structures

Parameters

- **onArray** (*list of dicts {trial, trialType, startTime, endTime, duration}*) – Gaze-on Center events
- **offArray** (*list of dicts {trial, trialType, startTime, endTime, duration}*) – Gaze-off events
- **trial** (*int*) – Trial number
- **ttype** (*string*) – Trial type
- **onArrayL** (*list of dicts {trial, trialType, startTime, endTime, duration}*) – Gaze-on Left events

- **onArrayR** (*list of dicts {trial, trialType, startTime, endTime, duration}*) – Gaze-on Right events
- **stimName** (*string*) – If presenting stimuli, name of the stim file
- **habTrialNo** (*int*) – Tracking if this is a habituation trial and if so what number
- **habCrit** (*float*) – Habituation criterion, if it's been set

Returns

Return type

dataRec(*onArray*, *offArray*, *trial*, *type*, *onArrayL*, *onArrayR*, *stimName=''*, *habTrialNo=0*, *habCrit=0.0*)

Records the data for a trial that ended normally.

Parameters

- **onArray** (*list of dicts {trial, trialType, startTime, endTime, duration}*) – Gaze-on Center events
- **offArray** (*list of dicts {trial, trialType, startTime, endTime, duration}*) – Gaze-off events
- **trial** (*int*) – Trial number
- **ttype** (*string*) – Trial type
- **onArrayL** (*list of dicts {trial, trialType, startTime, endTime, duration}*) – Gaze-on Left events
- **onArrayR** (*list of dicts {trial, trialType, startTime, endTime, duration}*) – Gaze-on Right events
- **stimName** (*string*) – If presenting stimuli, name of the stim file
- **habTrialNo** (*int*) – Tracking if this is a habituation trial and if so what number
- **habCrit** (*float*) – Habituation criterion, if it's been set

Returns

Return type

dispCoderWindow(*trialType=-1*)

Because there are three looking keys, the experimenter window has three boxes. Also things work differently with there being no 'secondkey' really. :param trialType: :type trialType: :return: :rtype:

dispTrial(*trialType*, *dispMovie=False*)

An HPP-specific version of dispTrial that can display on multiple things, read the new stimDict, etc.

Parameters

- **trialType** (*str*) – The trial type of the current trial being displayed
- **dispMovie** (*bool or dict*) – Now a dictionary C/L/R each index of which contains what this function expects in the base class

Returns 1 or 0. 1 = end of movie for trials that end on that. TODO: for HPP currently returns 1 if EVERYTHING IN IT is done.

Return type int

doTrial(*number*, *ttype*, *disMovie*)

Control function for individual trials, to be called by doExperiment Returns a status value (int) that tells

doExperiment what to do next HPP experiments works very differently from everything else, and this is where the bulk of that is happening.

Parameters

- **number** (*int*) – Trial number
- **ttype** (*string*) – Trial type
- **disMovie** (*dict*) – A dictionary with three indexes, one per screen. Any screen with stimuli on it will have a dictionary {stimType:,stim:}

Returns int, 0 = proceed to next trial, 1 = hab crit met, 2 = end experiment, 3 = trial aborted

Return type

endExperiment()
End experiment, save all data, calculate reliability if needed, close up shop :return: :rtype:

lookKeysPressed()
A simple boolean function to allow for more modularity with preferential looking Basically, allows you to set an arbitrary set of keys to start a trial once the attngetter has played. In this case, any of V, B, or N are sufficient.

Returns True if the V, B, or N key is pressed, False otherwise.

Return type

lookScreenKeyPressed(*screen=['C']*)
A function that primarily exists for HPP, because of the need to distinguish between any key being pressed and the key corresponding to the HPP screen in question being pressed

Parameters **screen** (*list of strings*) – List of screens for next stim.

Returns for non-HPP versions, the value of lookKeysPressed.

Return type bool

printCurrentData()
Prints current data to output window, HPP variant. Only called when stimulus presentation is off.

Returns

Return type

CHAPTER 7

Standalone Reliability

class StandaloneReliability

This script is simply the reliability function from *PyHab Class (base)* but run over two arbitrary verbose data files.

PyHab.**avgObsAgree**(*timewarp*, *timewarp2*)

Computes average observer agreement as agreement in each trial, divided by number of trials.

Parameters

- **timewarp** (*list*) – List of every individual frame's gaze-on/gaze-off code for coder A
- **timewarp2** (*list*) – As above for coder B

Returns average observer agreement or N/A if no valid data

Return type float

PyHab.**cohensKappa**(*timewarp*, *timewarp2*)

Computes Cohen's Kappa

Parameters

- **timewarp** (*list*) – List of every individual frame's gaze-on/gaze-off code for coder A
- **timewarp2** (*list*) – As above for coder B

Returns Kappa

Return type float

PyHab.**pearsonR**(*verboseMatrix*, *verboseMatrix2*)

Computes Pearson's R

Parameters

- **verboseMatrix** (*dict*) – Verbose data, coder A
- **verboseMatrix2** (*dict*) – Verboase data, coder B

Returns Pearson's R

Return type float

PyHab.**wPA**(*timewarp*, *timewarp2*)

Calculates weighted percentage agreement, computed as number of agreement frames over total frames.

Parameters

- **timewarp** (*list*) – List of every individual frame's gaze-on/gaze-off code for coder A
- **timewarp2** (*list*) – As above for coder B

Returns Weighted Percentage Agreement

Return type float

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Index

E

F

H

I

J

L

M

N

P

Q

R

S

T

U

W