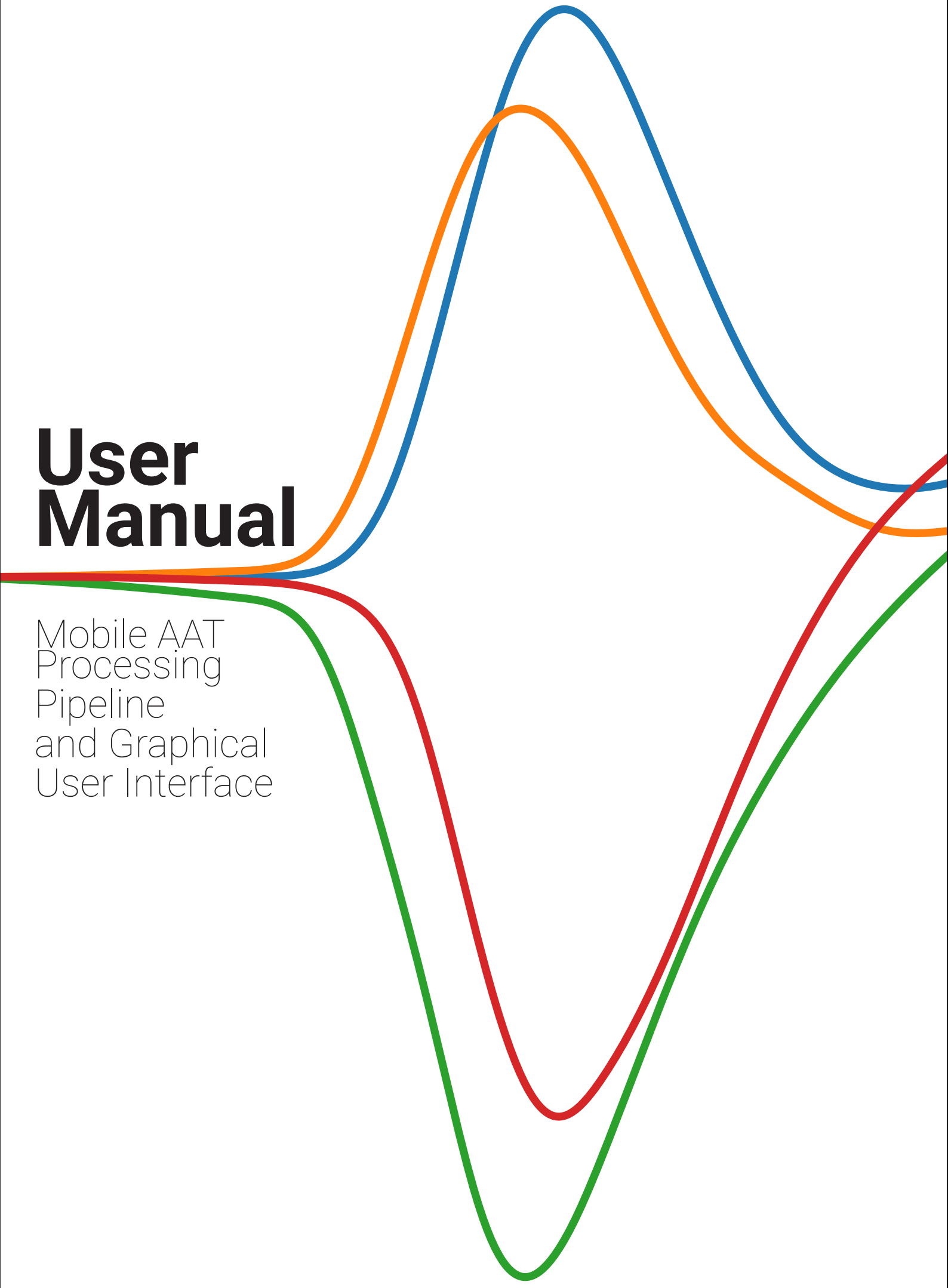
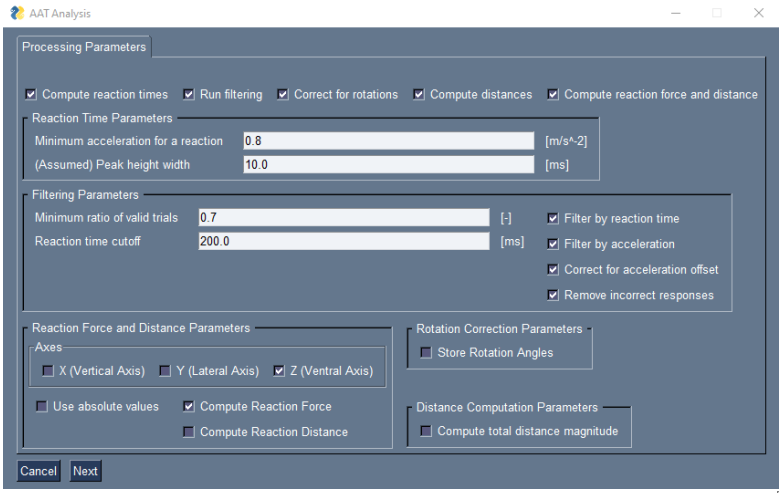
User Manual

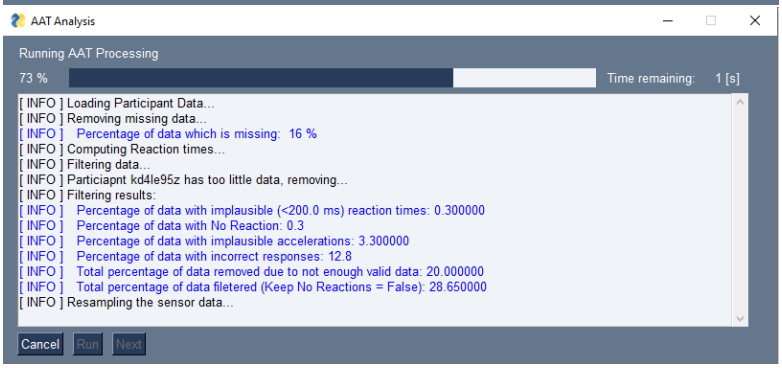
Mobile AAT Processing Pipeline and Graphical User Interface

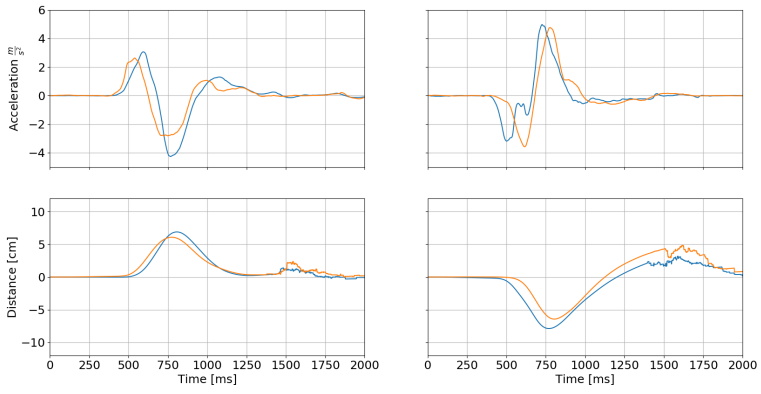


User Manual

Mobile AAT Processing Pipeline and Graphical User Interface







by

Jasper van Beers - jasper@vanbeers.dev
github.com/Jasper-van-beers

Contents

1 Introduction

The approach-avoidance task (AAT) can be used to measure implicit (approach-avoidance) tendencies and biases. The emergence of mobile variants of the AAT (mAAT), such as that of Zech et al. [1], enable the use of the AAT paradigm in-the-wild. To facilitate the ease of use of this tool, a Python-based processing pipeline compatible with the mAAT of [1] has been developed to process the raw data. The goal of this pipeline is to encourage the use of AAT paradigms for measuring (implicit) human behavioral aspects through an easy-to-use, open-source, processing tool. To make this tool accessible to all researchers, an accompanying intuitive graphical user interface (GUI) has also been developed to act as a frontend to the processing tool. Further, a plotting utility (also with a complementary GUI) has also been developed to allow for quick - and standardized - plotting of commonly used figures.

This document is written to give an overview of some of the functionalities of the processing pipeline and act as a user manual for the processing GUI and plotting GUI. Chapter 2 outlines the requirements of the pipeline and provides a guide for how to install the necessary Python dependencies. Chapter 3 and chapter 4 demonstrate how to use the processing GUI and plotting GUI, respectively. Moreover, an overview of the Data structures used by the pipeline are summarized in appendix A and appendix B.

The tool itself can be found on GitHub: `https://github.com/Jasper-van-beers/AAT`

2
Requirements & Installation

2.1 Requirements

This tool is written in Python (version 3.6.7) and as such requires Python (version 3.6.7 or later) to run. While not necessary, it is recommended that Anaconda is used to manage the Python environments. Anaconda can be downloaded from the following link:https://www.anaconda.com/products/individual

The Python library dependencies are provided in the `requirements.txt` file and can be installed as described in **Installing tool dependencies** in the subsequent section.

2.2 Installation

If Python, or the correct version (3.6.7 or later), is not installed then it can be found on the Python website: https://www.python.org/downloads/. The following steps outline how to install the tool.

Downloading and extracting the tool

1. The tool can be downloaded from GitHub: https://github.com/Jasper-van-beers/AAT
2. Once downloaded, please extract the files to the target directory

Installing tool dependencies: There are two methods to install the dependencies.

1. Using the supplied `InstallDependencies.py` file:
   (a) (Recommended) Open the anaconda console and navigate to the AAT directory (i.e. that which contains `InstallDependencies.py`). Type in the command: `python InstallDependencies.py`
   (b) Open the command prompt (or similar terminals from which python scripts can be run), navigate to the AAT directory (i.e. that which contains `InstallDependencies.py`). Then type in the command: `python InstallDependencies.py`
   (c) Double clicking the `InstallDependencies.py` file. Note that this only works if the correct version of Python (3.6.7 or later) is in your PATH variables.
2. Using `pip` to install the dependencies using the supplied `requirements.txt` file
   (a) (Recommended) Open the anaconda console, navigate to the AAT directory (i.e. that which contains `requirements.txt`) and type in the command: `pip install -r requirements.txt`
   (b) Open the command prompt (or similar terminals from which python scripts can be run), navigate to the AAT directory (i.e. that which contains `requirements.txt`). Then type in the command: `pip install -r requirements.txt`

3
Processing Pipeline GUI

This chapter outlines how to use the graphical user interface (GUI) of the processing pipeline. The various options available to users, and their subsequent results, of the GUI are presented here. The GUI itself is run from the `SimpleGUI.py` file, which is available in the AAT directory (see https://gituhub.com/Jasper-van-beers/AAT). To demonstrate the use of the GUI, data from Zech et al. (i.e. happy and angry faces) will be used [1]. These data are available at https://osf.io/y5b32/.

This chapter is structured by the various windows which users may encounter when using the GUI, wherein each section describes the purpose and functionality of the window in question.

Section 3.1 describes the first window users will see: the 'start window'. Depending on the user's choice on the start window, different windows will appear. Should users choose to import the raw data, then they will encounter the 'raw data importing window' that is described in section 3.2. If users have already run some processing (through the GUI) and would like to continue (e.g. to compute reaction distance), they may opt to 'run processing from file'. This window is presented in section 3.3.

Following the aforementioned windows, if users selected to run processing, then they will be prompted with the 'processing parameter selection window'. As discussed in section 3.4, this window is used to set the processing parameters (e.g. reaction time cutoff). After the processing parameters are set, an 'overview window' (presented in section 3.5) will be shown whereby a summary of the processing is given.

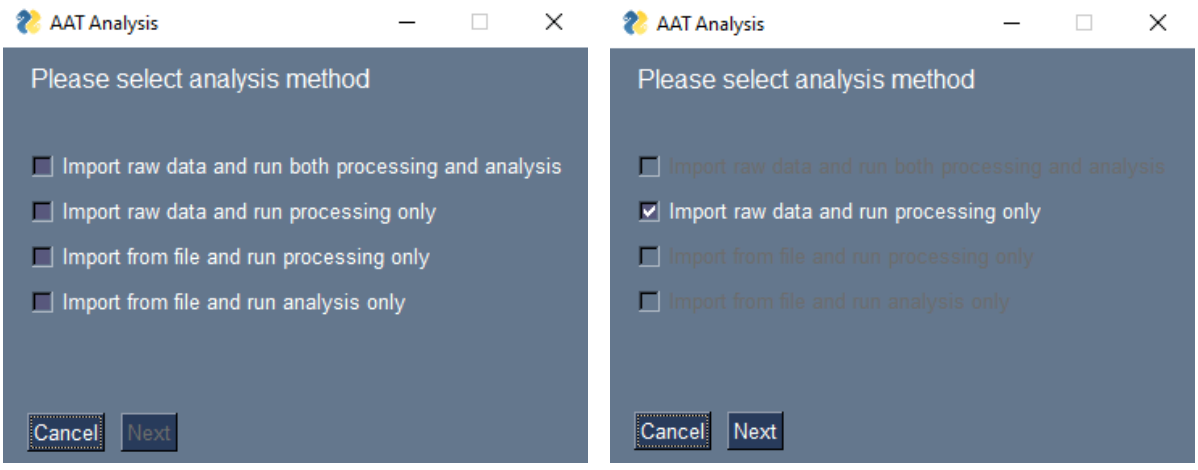
For users who selected to run the analysis, section 3.6 depicts the analysis parameter selection window. After the parameters have been selected, users are prompted with an 'analysis overview window' whereby as summary of the analysis steps is shown. This window is presented in section 3.7.

Moreover, to streamline the data processing, users are also given access to a GUI 'configuration file' which can pre-fill values of certain windows. This GUI configuration file is discussed in section 3.8.

3.1 Start Window

When users first run the program (`SimpleGUI.py`), they will be presented either fig. 3.1a, if no default is specified in `GUIConfig.json`, or with fig. 3.1b, where the user-specified default option is already pre-filled (see section 3.8 for more details on how to use the configuration file).

The start window itself allows users to indicate if they would like to run the processing and/or analysis from either the raw data or a (previously processed) file (see options in fig. 3.1a).



(a) All options

(b) Option selected

Figure 3.1: Overview of start window

Upon making a selection, the `Next` button becomes active and users may proceed with the option they have selected. While an option is selected, all other options are disabled (see fig. 3.1b). The program will be closed when users hit `Cancel` or the close button.

3.2 Raw Data Importing Window

Users who selected one of the 'import raw data and ...' options from section 3.1 are subsequently presented with fig. 3.2. In this window, users may browse for the relevant raw data folders. These are:

- "Please locate the raw data folder" - The path to the raw data files (typically located under `AAT\data\raw`)
- "Please locate the conditions fodler" - The path to the 'conditions folder' where the `conditions.json`, `sessions.json`, `stimulus_sets.json`, and `tasks.json` (and `blocks.json` for newer versions of the mAAT) are located (typically under `AAT\data\external`).

Users may also indicate whether they would like to save the processed file. If so, options are also available to specify the output directory, filename and filetype (.pkl or .csv). Note that if users want to conduct additional processing on a (partially) processed save file, the `Save condensed data` option **must** be unchecked. This will save the output as a .pkl file (note: this extension should not be added to the `Save filename`). Checking the `Save condensed data` option instructs the program to only save the necessary variables for statistical analyses, and data needed to do further processing is therefore lost. Moreover, the `Save as .csv` option is only available when `Save condensed data` is checked since the 'uncondensed' data is not compatible with the .csv format.

All entries in fig. 3.2 can be pre-filled using the configuration file (see section 3.8).

Figure 3.2: Overview of the window used to run processing/analysis from the raw mAAT data

Once users have filled in the window entries, the 'Next' button becomes available and, when clicked, users will be directed to the processing parameter selection window (see section 3.4). Note that the 'Next' button only becomes active if the entered paths (i.e. directories) actually exist, so if the 'Next' button is unavailable, please check the spelling of the directories or use the folder browser to locate the relevant directory.

3.3 Run Processing from File Window

Figure 3.3 depicts the window presented to users who selected one of the 'import from file ...' options in section 3.1. Here, users can use the file browser to locate the (.pkl) file they would like to run additional processing or analysis on. The saving options are the same as with section 3.2 wherein users can indicate the save directory, save filename and output filetype (.pkl or .csv). For more details about the saving options, please refer to section 3.2.

Attention: when importing from file, it important that the imported file has the relevant metadata associated with that file. When saving files with the GUI, the program also creates a metadata file. For example,

with a save file called `MyAATData.pkl` the associated metadata file is called `MyAATData_metadata.json` and can be found in the same directory as the save file. This metadata file is necessary to run additional processing/analysis. Therefore, please ensure that the metadata file is present in the same directory as the target .pkl file. If the target .pkl file is renamed, also rename the metadata file accordingly. The GUI will alert the user if the metadata file cannot be found.

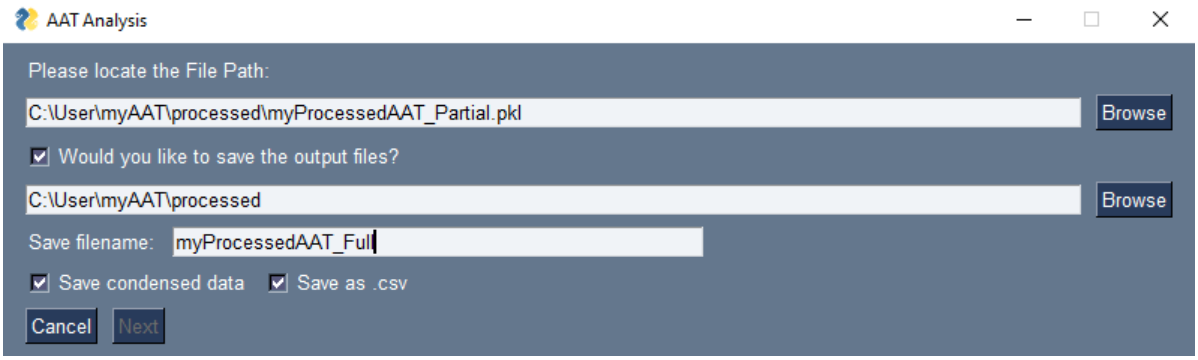


Figure 3.3: Overview of the window used to run processing/analysis from a previously processed mAAT data file.

To proceed with the processing/analysis, users can click the 'Next' button. However, this button is only activated when the provided .pkl file exists. If the 'Next' button is unavailable, please check the spelling of the file paths or use the file browser to locate the relevant file.

Depending on the size of the file, it may take some time to load. While the file is loading, users are prompted with fig. 3.4. Once the file has loaded, fig. 3.4 will automatically close.

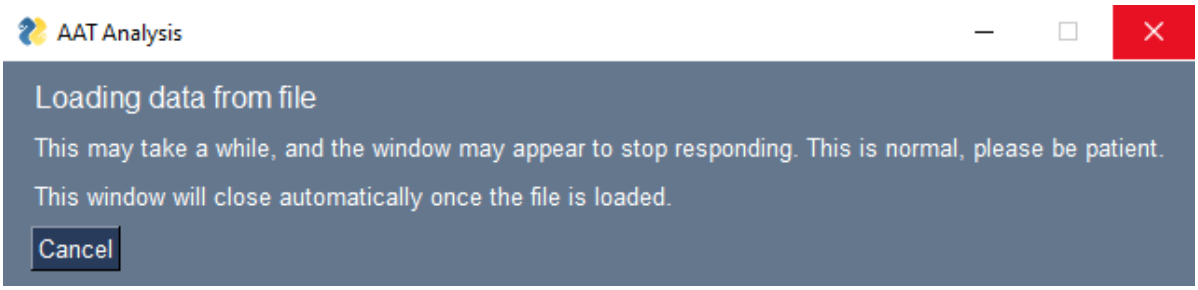


Figure 3.4: Overview of the window when the tool is busy loading a file.

3.4 Processing Parameter Selection Window

Before the processing can be run, the processing parameters need to be specified. This is facilitated through the window depicted in fig. 3.5. The processing parameters can be pre-filled using the configuration file described in section 3.8. If no configuration file is used, then the default parameters are pre-selected. Figure 3.5 is organized into five different processing categories which can each be toggled through the checkboxes along the top of the window. For explanations on what each parameter controls readers are directed to section 3.8.4 which summarizes the parameters, per processing category. Users should take special care to ensure that the inputs in the field entries (e.g. Reaction time cutoff) are in the correct data type, as indicated in section 3.8.4.

Some processing categories depend on the results of others. For example, in order to compute the distances, users must first filter the raw data. The dependencies are ingrained in the GUI, hence, the program will notify users if certain dependencies are not met when they hit the 'Next' button. Moreover, the error will specify which dependencies are necessary to run certain functions. In general, most functions require the AAT data to be filtered.

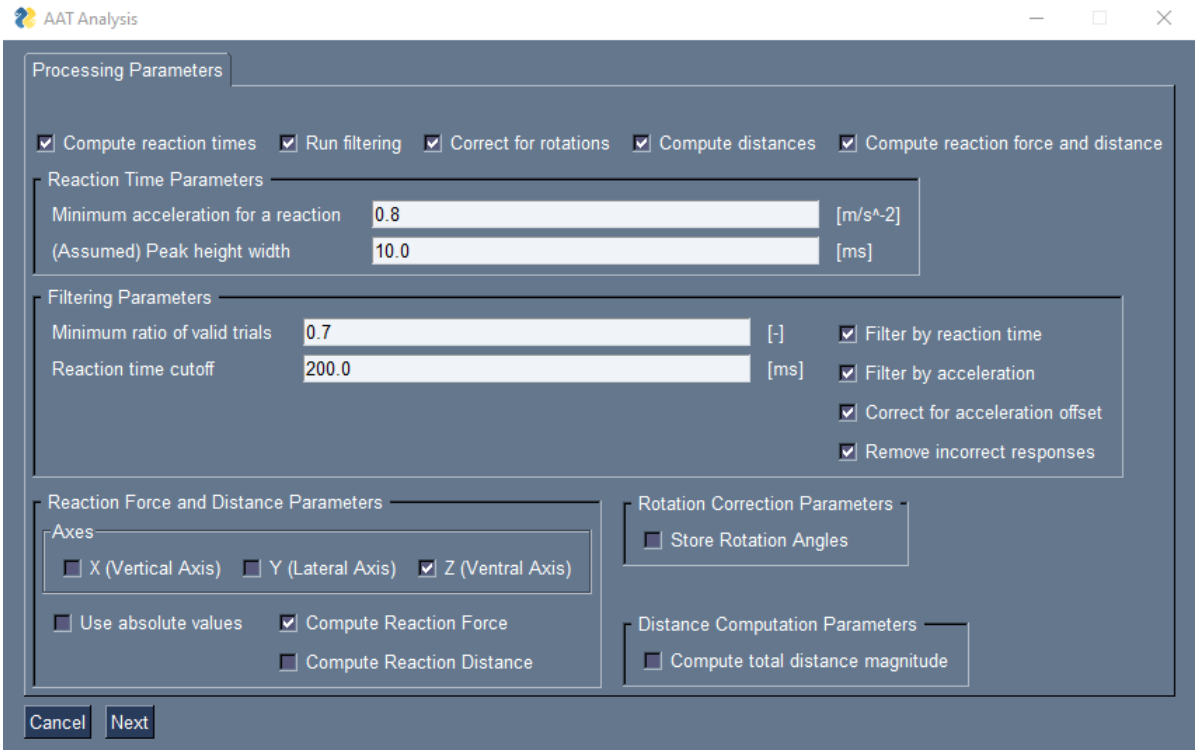


Figure 3.5: Overview of the window used to specify the processing parameters.

3.5 Processing Overview Window

Once users are satisfied with their processing parameters, and all dependency requirements are met, the processing can be run through fig. 3.6. **Note** that the processing will not commence until users hit 'Run'. While the data is being processed, fig. 3.6 relays some of the processing information to the user, such as the current processing step and results (e.g. percentage of participants with incorrect responses). These results are printed in blue to distinguish them from general information (printed in black). Once the processing is done, the 'Next' button becomes active. The program can be cancelled at any time by hitting the 'Cancel' button.

Figure 3.6: Overview of the window that summarizes the current state of the processing.

Attention: If users encounter an out-of-memory (OOM) error, then it is likely the case that the processed

data is taking up too much RAM. To verify if this is the case, users can check the memory use of Python while running the preprocessing. If OOM errors occur, users can try split up the raw data into batches and run the processing on these batches. The data between batches can then be combined after the data has been processed. For additional information on what parameters can inflate memory use, see section 3.8.4. If memory issues persist, users can manually remove unnecessary columns from their DataFrames (see appendices A and B for all available columns and their descriptions).

3.6 Analysis Parameter Selection Window

Figure 3.7 depicts the analysis parameter specification window. This window is divided into the Stimuli, Averaging Options, and Statistics sections.

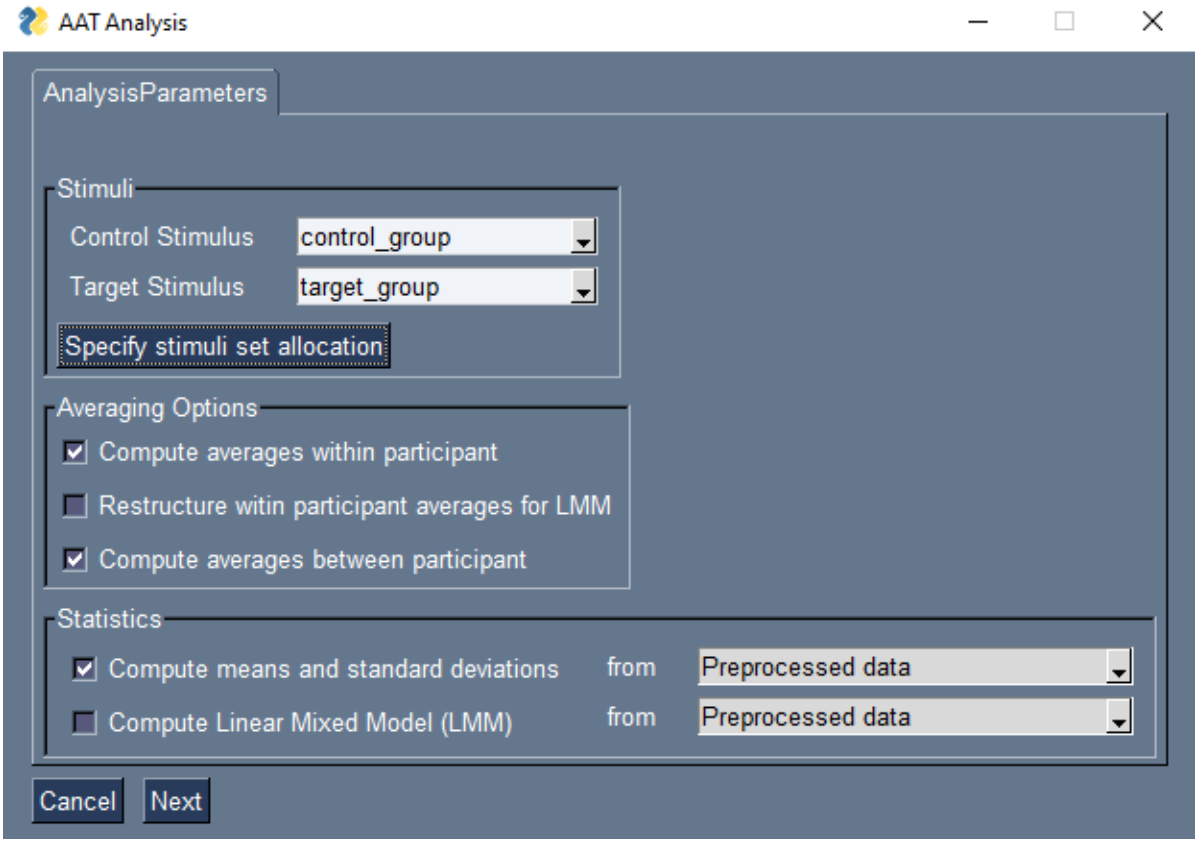


Figure 3.7: Overview of the window used to specify the analysis parameters.

Before any analysis can be done, users need to specify the `Control` and `Target` stimuli. The analysis is structured such that the 'congruent' condition (i.e. the null hypothesis of the statistical analysis) is `Pull x Target` (i.e. pulling the target stimulus results in faster reaction times, higher reaction forces and higher reaction distances). Users can either type their own names or select one from the dropdown menu (see Figure 3.8). The stimuli in the dropdown menu are derived from the stimulus set names contained in the mAAT (raw) data.

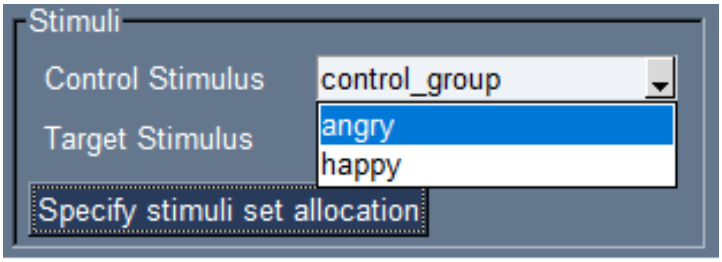


Figure 3.8: Example of (automatically) generated control and target group allocation. User specification is possible

Once the control and target stimuli are specified, users need to `Specify stimuli set allocation`. In essence, this is where users map the different conditions of the experiment to either the control or target stimulus. This is necessary as it informs the program how to group the data stemming from different sessions and (subsets of) stimuli. An example of this mapping is given in fig. 3.9. In the given example, the allocation is rather trivial in that, for each of the possible sessions, data from the happy stimulus sets are grouped into the 'happy group' while data from the angry stimulus sets are grouped into the 'angry group'. However, for more complex mAAT paradigms, it may be the case that a given stimulus set has further subsets (e.g. happy_a, happy_b etc.) which may either need to be grouped together (e.g. into happy_ab) or into individual groups (e.g. set_a, set_b).

If there are not enough entries available for the required mapping, more entries can be added through the 'Add more entries' button. This will open an identical window to fig. 3.9. Once users are happy with their mapping, they can hit confirm to continue with the analysis. It is important that at there is data allocated to both the target and control stimuli. If this is not the case, users are unable to confirm their mapping. **Note** that this error is only checked at the end of stimulus mapping, and will not be checked for any child windows created using the `Add more entries` button.

Furthermore, the program saves a 'group mapping' excel file (`<Filename>_GroupMapping.xlsx`; where `<Filename>` is the specified Save Filename from section 3.2 or section 3.3) which visualizes the mapping. This serves two objectives: the first is to verify that the selected mapping is indeed correct and the second is to serve as a reminder for what the mapping is for a given processed dataset.

Figure 3.9: Example of tool used to map sessions and stimulus sets into the user-specified control and target groups

After the mapping has been confirmed, users are brought back to the main analysis parameter window (fig. 3.7). Here, the `Averaging Options` may be specified. By default, the within participant averages are computed. This entails averaging all the trials, per participant, under the following four conditions for each of the selected metrics (i.e. Reaction time, Reaction Force and/or Reaction distance):

- Pull x Target

- Push x Control
- Pull x Control
- Push x Target

Users are also given the option to compute the between participant averaged data (see `Compute averages between participant` in fig. 3.7), which averages the data across the same four conditions and across all participants. While this overall average is perhaps not so informative for statistics, it provides a nice overview for the average responses (e.g. acceleration-time) of the participants under a given condition.

By default, the statistical analyses are computed on the processed data (and not the within participant averaged data). However, the statistical analysis can also be run on the within participant averaged data if it is restructured to do so. To do this users must first select `Restructure within participant averages for LMM` in fig. 3.7 and then users must indicate that the LMM should be computed from `Within participant averaged data` as highlighted in the dropdown menu in fig. 3.10.

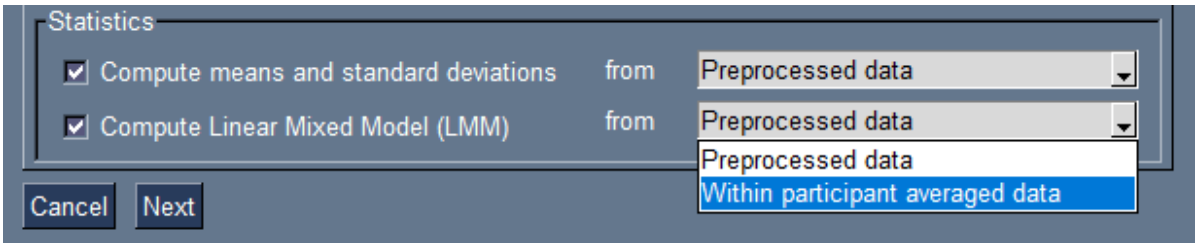


Figure 3.10: Options for built-in statistical analyses

3.7 Analysis Overview Window

Following the analysis parameter selection, users are presented with an overview window (fig. 3.11) similar to fig. 3.6. Again, the 'Run' button must be clicked to start the analysis. Once running, the current state of the analysis, and some results, are displayed on the output section.

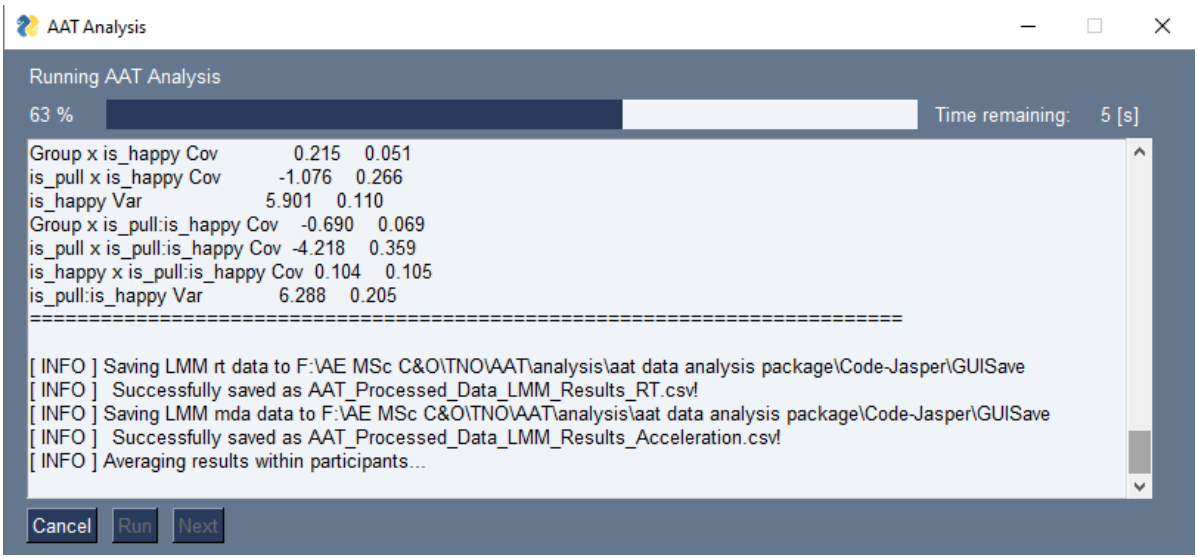


Figure 3.11: Overview of the window that summarizes the current state of the analysis.

3.8 GUI configuration file

This section describes how the configuration file (called `GUIConfig.json`) may be used alongside the GUI to pre-fill, and otherwise streamline, the processing/analysis process. This configuration file may be opened using notepad or its other variants. To use this file, it must be located in the same directory as the GUI file (i.e. `SimpleGUI.py`).

The structure of the `GUIConfig.json` mirrors that of the program, in that, each 'section' of the configuration file corresponds to a window of the GUI.

3.8.1. GUIConfig: General options

The general options of the configuration file are managed by lines 2-3 in listing 3.1 (found in section 3.8.6). The parameters control the following:

Parameter	Type	Description
`UseConfig`	Boolean	Toggles the use of the configuration file; true = use GUIConfig
`Analysis Method`	String	Pre-fills analysis method in 'start window' (section 3.1). The number corresponds to the analysis method in fig. 3.1 from top ("0") to bottom ("3"). For example, 'import raw data and run processing only' is specified with "1"

3.8.2. GUIConfig: Run Processing from Raw Data Options

This section provides the pre-fillable parameters for users who wish to run some processing directly from the raw data. In `GUIConfig.json`, these parameters are specified in lines 5-11 in listing 3.1 (i.e. under `Processing Directories`). The parameters control the following:

Parameter	Type	Description
`Raw Data Path`	String	Path to the (by default) 'raw' directory. This directory contains all the participant raw data. **Note:** the path should be specified using a double backslash (i.e. \\)
`Conditions File Path`	String	Path to the (by default) 'external' directory. This directory contains the files pertaining to the setup of the mAAT (i.e. has files 'conditions.json', 'sessions.json', 'stimulus_sets.json' etc.). **Note:** the path should be specified using a double backslash (i.e. \\)
`Save File`	Boolean	Toggles if the resultant processed file should be saved or not. Set to `true` to save the resultant file.
`Save Filename`	String	The filename of the processed data file.
`Save Path`	String	Output directory of the processed save file. **Note:** the path should be specified using a double backslash (i.e. \\)
`Save Condensed`	Boolean	Toggles if only the essential information (for analysis; i.e. trial information, Reaction Time, Reaction Force, and Reaction Distance) should be saved. If set to `false`, then the full DataFrame (e.g. acceleration responses of each trial, along each axis) will be saved.
`Save as .csv`	Boolean	Toggles if the condensed data should be saved as a .csv file or a .pkl file. If `true`, then the condensed data is saved to a .csv file. **Note:** saving to a .csv file is only possible if the condensed data is saved.

3.8.3. GUIConfig: Run Processing from File Options

This section provides the pre-fillable parameters for continuing the processing/analysis from a saved (partially) processed data file. Note that this option only works for processed (only .pkl and not .csv) files previously created through the GUI. In `GUIConfig.json`, these parameters are specified in lines 14-19 in listing 3.1 (i.e. under `Import File Information`). The parameters control the following:

Parameter	Type	Description
`File Path`	String	Path to the processed data .pkl file. **Note:** the path should be specified using a double backslash (i.e. \\)
`Save File`	Boolean	Toggles if the resultant processed file should be saved or not. Set to `true` to save the resultant file.
`Save Filename`	String	The filename of the processed data file.
`Save Path`	String	Output directory of the processed save file. **Note:** the path should be specified using a double backslash (i.e. \\)
`Save Condensed*`	Boolean	Toggles if only the essential information (for analysis; i.e. trial information, Reaction Time, Reaction Force, and Reaction Distance) should be saved. If set to `false`, then the full DataFrame (e.g. acceleration responses of each trial, along each axis) will be saved.
`Save as .csv*`	Boolean	Toggles if the condensed data should be saved as a .csv file or a .pkl file. If `true`, then the condensed data is saved to a .csv file. **Note:** saving to a .csv file is only possible if the condensed data is saved.

*Note: These parameters are only relevant if users are running processing and are ignored when users are running only the analysis.

3.8.4. GUIConfig: Processing Parameter Options

This section outlines the pre-fillable parameters used for the processing of the raw or partially processed AAT data. In `GUIConfig.json`, these parameters are specified in lines 22-51 in listing 3.1 (i.e. under `Default Processing Parameters`). These parameters are further split by processing category (for example, `Filtering Parameters`) for better clarity. The description of the parameters is therefore also organized as such.

Filtering Parameters

The filtering parameters can be found on lines 23-29 in listing 3.1 and represent the filtering options available to the user.

Parameter	Type	Description
`Run`	Boolean	Toggle the filtering of the AAT data. If `true` then the data will be filtered.
`Minimum ratio of valid trials`	Float	Describes the minimum amount of valid trials, as a ratio of the total number of experiment trials, required per participant. If participants have fewer valid trials, all their data will be discarded.
`Reaction time cutoff`	Int	Minimum reaction time, in milliseconds, for a valid (i.e. realistic) response. Ignored if `Filter by reaction time` is `false`.
`Filter by reaction time`	Boolean	Toggle filtering by reaction time. Filtering is applied if `true`.
`Filter by acceleration`	Boolean	Toggle filtering by acceleration. Filtering is applied (i.e. unrealistic accelerations removed) if `true`.
`Correct for acceleration offset`	Boolean	Toggle the correction for sensor bias/offset, during baseline period, in accelerometer data. Correction is applied if `true`.
`Remove incorrect responses`	Boolean	Toggle the removal of incorrect responses. If `true`, then incorrect responses will be removed from the processed data.

Reaction Time Parameters

The reaction time parameters can be found on lines 32-34 in listing 3.1 and represent the reaction time related options available to the user.

Parameter	Type	Description
`Run`	Boolean	Toggle the computation of reaction times of the AAT data. If `true` then the reaction times will be calculated.
`Minimum acceleration for a reaction`	Float	Minimum acceleration (along the approach-avoidance axis) in $m \cdot s^{-2}$ that is required for a response to be considered a reaction.
`(Assumed) Peak height width`	Int	Minimum allowable temporal distance (in milliseconds) between two (acceleration) response peaks

Rotation Correction Parameters
User-specifiable parameters related to the rotation correction processing can be found in lines 37-38 of listing 3.1.

Parameter	Type	Description
`Run`	Boolean	Toggle the correction for rotations of the phone during a response. If `true` then the rotations will be corrected for.
`Store Rotation Angles`	Boolean	Toggle if the derived rotation angles should be stored in the processed DataFrame. If `true`, then rotation angles are kept. **Note:** this significantly adds to the file size of the DataFrame, so this option is not recommend for users facing out-of-memory problems

Distance Computation Parameters
User-specifiable parameters related to the derivation of distance responses can be found in lines 41-42 of listing 3.1.

Parameter	Type	Description
`Run`	Boolean	Toggle the computation of distance responses. If `true` then distances will be derived from the accelerations.
`Compute total distance magnitude`	Boolean	Specify if the total distance (i.e. combination along all axes) should also be computed. **Note:** This is not recommended since the distance estimates outside the principle axis of motion (i.e. approach-avoidance) are typically unreliable. Moreover, this adds to the size of the resultant DataFrame. If `true`, then the distance magnitude will be computed.

Reaction Force and Distance Parameters
User-specifiable parameters related to the computation of the Reaction Force (RF) and Reaction Distance (RD) can be found in lines 45-51 of listing 3.1.

Parameter	Type	Description
`Run`	Boolean	Toggle the computation of RF and/or RD. If `true` then RF and/or RD will be calculated.
`X (Vertical Axis)*`	Boolean	Toggle the computation of RF/RD along the x-axis. If `true` then RF/RD will be computed along this axis.
`Y (Lateral Axis)*`	Boolean	Toggle the computation of RF/RD along the y-axis. If `true` then RF/RD will be computed along this axis.
`Z (Ventral Axis)*`	Boolean	Toggle the computation of RF/RD along the z-axis. If `true` then RF/RD will be computed along this axis.
`Use absolute values`	Boolean	Specify if the absolute value of the RF/RD should be taken. For 'push' (i.e. avoidance) movements, the resultant RF/RD values are negative. This results in a bi-modal distribution, which may be incompatible with certain analysis methods. Hence, the absolute value can be taken to transform the 'push' distribution to positive values. This is typically advised, but not strictly necessary. If `true`, then the absolute values will be taken.
`Compute reaction forces`	Boolean	Toggle the computation of RF. If `true`, RF will be computed.
`Compute reaction distances`	Boolean	Toggle the computation of RD. If `true`, RD will be computed.

*__Note:__ If multiple axes are selected, then the RF/RD will be computed across all selected axes (i.e. the single RF/RD value will be representative of the RF/RD over all the axes). Due to this, it is recommended to only compute the RF/RD in the direction of the principle motion (for the traditional mAAT setup, this is along the z-axis).

3.8.5. GUIConfig: Analysis Parameter Options

The user-specifiable analysis parameters can be found under `Analysis Parameters` (lines 54-64 in listing 3.1) as with the processing parameters, the analysis parameters are also organized into further subgroups: general parameters, averaging options, and statistics.

General parameters

It is important for the analysis steps to specify which stimulus is the 'control' stimulus and which is the 'target' stimulus. The mAAT tool is setup such that the congruent condition is Pull x Target and Push x Control while the incongruent condition is Push x Target and Pull x Control. The control and target stimuli can be specified through lines 55-56 in listing 3.1. In the analysis parameter selection window, users can specify the mapping for stimulus sets to either the control or target stimulus (refer to section 3.6 for more details).

Parameter	Type	Description
`Control`	String	Label for the control stimulus (where: Push x Control is Congruent)
`Target`	String	Label for the target stimulus (where: Pull x Target is Congruent)

Averaging Options

In this subgroup, users may specify how they want to average the data. In `GUIConfig.json`, these parameters can be found on lines 58-60 (see listing 3.1).

Parameter	Type	Description
`Compute within participant averages`	Boolean	Dictates if the data should be averaged within participants (i.e. per participant, averages across the individual trials are taken for each of the conditions: Pull x Target, Push x Target, Pull x Control and Push x Control). If `true`, then the within-participant averages are taken.
`Restructure averaged data for LMM*`	Boolean	The Linear Mixed Models (LMM) can be run on directly on the processed DataFrame, or on the within-participant averaged data. However, the within-participant output DataFrame is not compatible with the LMM scripts, hence, this parameter - if `true` - restructures said DataFrame to make it compatible for LMM computations.
`Compute between participant averages*`	Boolean	Dictates if the data should be averaged between participants (i.e. per condition: Pull x Target, Push x Target, Pull x Control and Push x Control, responses are averaged across all participants and trials). If `true`, then the between-participant averages are taken.

***Note:** These options can only be selected if the `Compute within participant averages` option is also selected.

3.8.6. GUIConfig: The file

Listing 3.1 gives an example of a filled in configuration file.

Listing 3.1: (Example) Parameters of GUIConfig.json

```
{
    "UseConfig":true,
    "Analysis Method" : "0",
    "Processing Directories": {
        "Raw Data Path":"Path\\to\\raw\\data\\folder",
        "Conditions File Path":"Path\\to\\external\\data\\folder",
        "Save File":true,
        "Save Filename":"myProcessedAAT",
        "Save Path":"Path\\to\\save\\folder",
        "Save Condensed":true,
        "Save as .csv":true
    },
    "Import File Information": {
        "File Path": "Path\\to\\AAT\\pkl\\file",
        "Save File":true,
        "Save Filename":"myProcessedAAT",
        "Save Path":"Path\\to\\save\\folder",
        "Save Condensed":true,
        "Save as .csv":true
    },
    "Default Processing Parameters": {
        "Filtering Parameters": {
            "Run":true,
            "Minimum ratio of valid trials":"0.75",
            "Reaction time cutoff":"200",
            "Filter by reaction time":true,
            "Filter by acceleration":true,
            "Correct for acceleration offset":true,
            "Remove incorrect responses":true
        },
        "Reaction Time Parameters": {
            "Run":true,
            "Minimum acceleration for a reaction":"0.8",
```

```
            "(Assumed) Peak height width":"10"
        },
        "Rotation Correction Parameters": {
            "Run":true,
            "Store Rotation Angles":false
        },
        "Distance Computation Parameters": {
            "Run":true,
            "Compute total distance magnitude":false
        },
        "Reaction Force and Distance Parameters": {
            "Run":true,
            "X (Vertical Axis)":false,
            "Y (Lateral Axis)":false,
            "Z (Ventral Axis)":true,
            "Use absolute values":false,
            "Compute reaction forces":true,
            "Compute reaction distances":false
        }
    },
    "Analysis Parameters": {
        "Control":"control_group",
        "Target":"target_group",
        "Averaging Options": {
            "Compute within participant averages":true,
            "Restructure averaged data for LMM":false,
            "Compute between participant averages":true
        },
        "Statistics": {
            "Compute means and standard deviations":true,
            "Compute Linear Mixed Models":false
        }
    }
}
```

4
Plotter GUI

This chapter outlines how to use the GUI plotting utility bundled with the mAAT processing tool. The idea behind this GUI plotting utility is that it allows for the easy, yet flexible, plotting of commonly used plots (such as Acceleration-Time). Moreover, multiple different plots can be put in the same figure. The exact configuration is up to the user. The GUI itself is run from the `GUIPlotter.py` file, which is available in the AAT directory (see https://gituhub.com/Jasper-van-beers/AAT). To demonstrate the use of the GUI, data from Zech et al. (i.e. happy and angry faces) will be used [1]. These data are available at https://osf.io/y5b32/.

4.1 File Importer

When run, the `GUIPlotter` will first ask users to specify an import file. This file can be found through the accompanying file browser (see fig. 4.1). This file can either be the processed DataFrame (with structure similar to that outlined in appendix A) or the averaged DataFrame (with structure similar to that summarized in appendix B). Only .pkl files are compatible with the plotting utility. The "Next" button only becomes active if the provided file exists.

Figure 4.1: `GUIPlotter` file importing window.

Depending on the file size, it may take some time to load the file. During the loading process, users will be presented with fig. 4.2. Once the file has loaded, the window will automatically close.

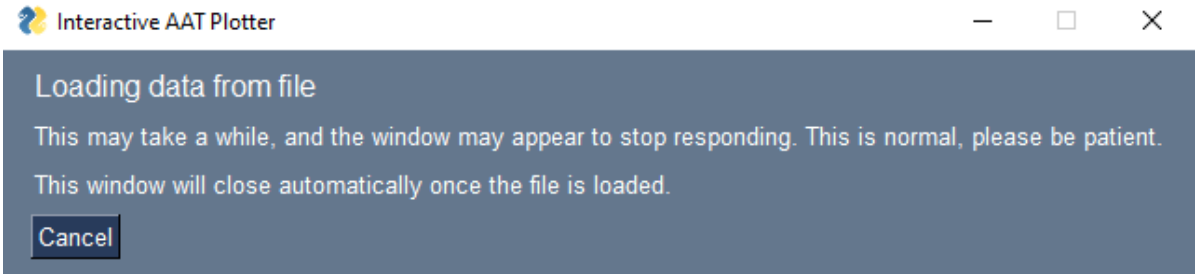


Figure 4.2: `GUIPlotter` file loading window.

4.2 Plotting Window

After the file has been loaded, fig. 4.3 will pop-up. This is the default 'plotting window', where only one plot will be shown in the figure when the `Plot` button is hit. To customize the number of plots in a figure, users can click the `Settings` button. Doing so will prompt fig. 4.4, wherein users can design their own plot layout by specifying the number of rows and columns of the figure. Each entry in this grid can host a plot, or may be left empty (if `None` is plotted). In fig. 4.4, a 2 x 2 grid is specified. This results in four plotting regions of equal size.

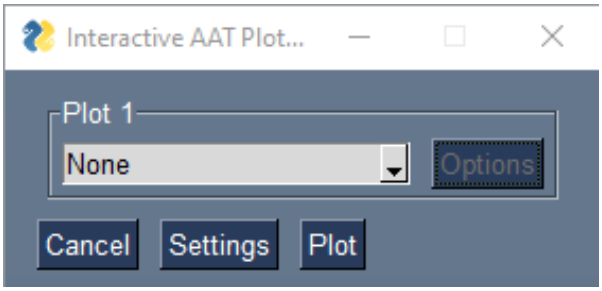


Figure 4.3: `GUIPlotter` 1 x 1 plotting window.

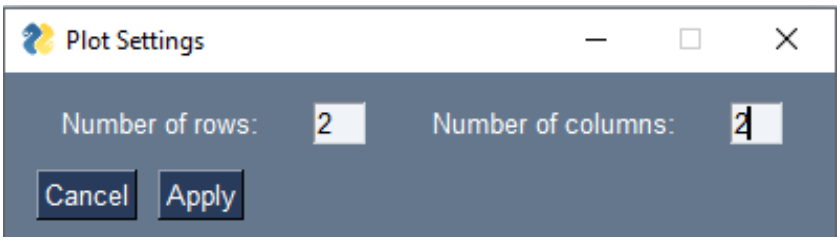


Figure 4.4: `GUIPlotter` plot layout design window.

Continuing with the 2 x 2 layout example, fig. 4.5 depicts the resulting plotting window.

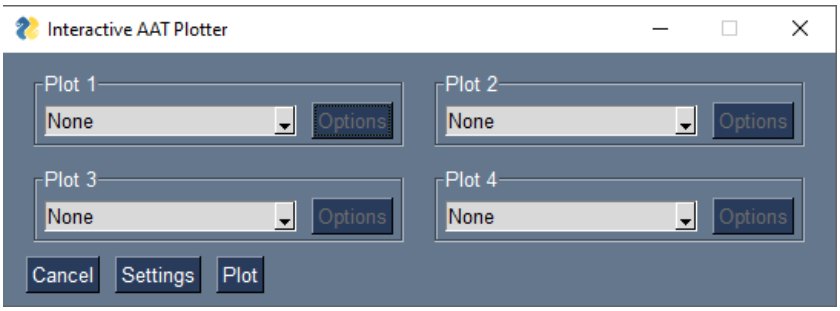


Figure 4.5: `GUIPlotter` 2 x 2 plotting window.

The available plots can be found by clicking the dropdown menu (see fig. 4.6). A description of these options are summarized in table 4.1.

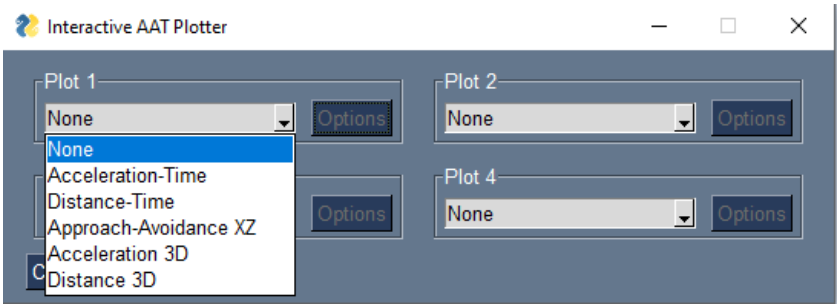


Figure 4.6: `GUIPlotter` Available plots in the dropdown menu

Table 4.1: Summary of the available plots in the `GUIPlotter` plotting utility

Plot	Description
`None`	Empty plot region
`Acceleration-Time`	Acceleration response, as a function of time
`Distance-Time`	Distance response, as a function of time
`Approach-Avoidance XZ`	Acceleration or Distance response, as seen in the ventral and vertical axis (i.e. approach-avoidances and up-down axis)
`Acceleration 3D`	Acceleration response along all 3-axis of motion. Time is encoded by starting and stopping points
`Distance 3D`	Distance response along all 3-axis of motion. Time is encoded by starting and stopping points

Once users have selected a plot, the `Options` button becomes active. Clicking this button will open the plotting options, depicted by fig. 4.7, wherein they can specify how the plot is to be constructed. The parameters, and their descriptions, are summarized in table 4.2. Users need to click `Apply` to confirm the plotting options.

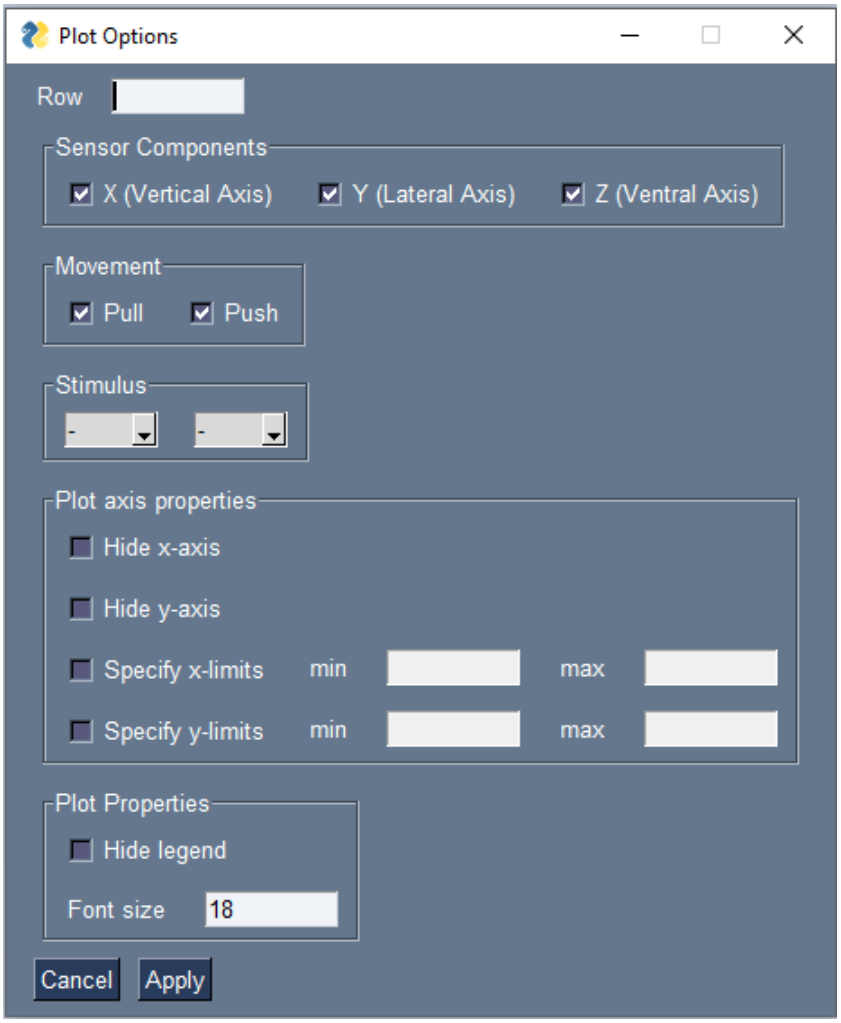


Figure 4.7: `GUIPlotter` Plotting options (example used for Acceleration-Time and Distance-Time, other options are similar)

Table 4.2: Summary of plotting options outlined in fig. 4.7

Option	Description
`Row`	The row - starting from 0 - in the DataFrame which data should be taken from. Depending on the DataFrame, this could correspond to an individual trial (Processed DataFrame), an individual participant (Within-participant averaged DataFrame) or to the overall average (Between-participant averaged DataFrame).
`Sensor Components`	The axis components which should be plotted. For most plots, only the Z-axis is required (in-line with the approach-avoidance axis)
`Movement`	The movement (Push, Pull, or Both) which should be plotted.
`Stimulus`	The stimuli that should be plotted. Users must select a stimulus from the dropdown menu, the available stimuli are derived from the imported DataFrame.
`Plot axis properties`	Some properties pertaining to the presentation of the plot axes. Users can hide (either) the x- and y-axes, and are able to specify the limits for these axes. The limits should be inputted as integers or floats.
`Plot Properties`	Some general properties which the users can specify for the plot. These include hiding the (plot-specific) legend and specifying the font size for all plot labels.

Once the plot options are applied, users can see an overview of the inputted options by hovering over the plot region (see fig. 4.8). In the event that users did not specify any plot options, then the overview will show the message 'No data'. This is done to provide users with an overview of what options they have selected and for which plots, and to help users keep tabs on what plots have been constructed and what plots are lacking data.

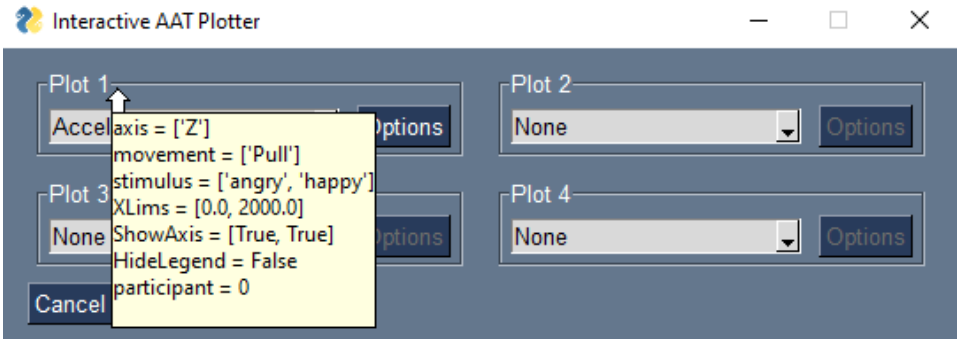


Figure 4.8: `GUIPlotter` Summary of applied plot options. Summary is shown when users hover their mouse over the plot region (in this example, 'Plot 1')

Figure 4.9 illustrates an example plot, using the between-participant averaged happy-angry data from [1], created using the `GUIPlotter`.

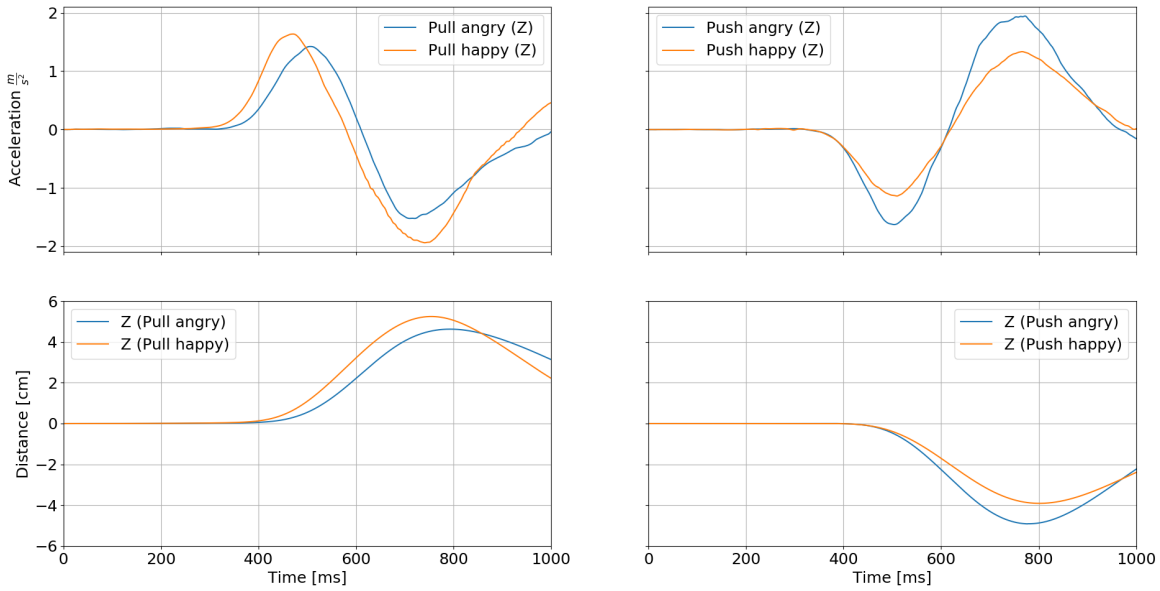


Figure 4.9: Example plot created using `GUIPlotter`. Here, a 2x2 plot is made with the top row being acceleration-time graphs and the bottom row distance-time graphs. The axes have been set to be equal and certain axes are hidden for aesthetics.

Bibliography

[1] Hilmar G. Zech, Mark Rotteveel, Wilco W. van Dijk, and Lotte F. van Dillen. A mobile approach-avoidance task. Behavior Research Methods, 2020.

A

Appendix A: Processed DataFrame Structure

This appendix provides some documentation on the structure of the DataFrame used to hold the (partially) processed AAT data. Since the rows to the DataFrame correspond to the participants and trials, they will not be discussed further here. The main focus of this appendix is to provide an overview of the columns of the DataFrame and the data they hold. The various DataFrame columns, and their accompanying descriptions, added by different functions are summarized in tables A.1 to A.8.
ATTENTION:

- Variables with * are modified by other functions. Depending on where in processing pipeline you are, these variables could be representative of slightly different things. Please use the most up-to-date description of the variable (i.e. the description corresponding to the most recently used function, e.g. `ResampleData` modifies the accelerations).

- For mAAT designs where a questionnaire is asked on the app (e.g. age, gender, height etc.) the corresponding columns will be added by `ImportData` to the DataFrame. These are not shown in table A.1 since the column names depend on the labels assigned in the mAAT app itself. Hence, to get an overview of these columns, users can call `DataFrame.columns`, where "DataFrame" is the variable name of the DataFrame containing the mAAT data. For users of the GUI, these columns are also automatically added to the condensed version of the processed data.

Table A.1: DataFrame columns created by function `ImportData()`

Column Name	Description
participant	AAT ID of the participant, assigned by mAAT
condition	Experimental condition of participant
session	Current session of the trial (e.g. push_happy_session)
block	Current block number of the trial
block_trial	Current trial number, within a block
trial	Current trial number, within a session
_globalTrials	Current total trial number
acceleration*	Acceleration along z-axis (approach-avoidance/ventral axis)
acceleration_x*	Acceleration along x-axis (vertical axis)
acceleration_y*	Acceleration along y-axis (lateral axis)
times*	Time corresponding to accelerometer data
gyro_times	Time corresponding to gyroscopic data, if available
gyro_x*	Angular rate about x-axis (vertical axis), if available
gyro_y*	Angular rate about y-axis (lateral axis), if available
gyro_z*	Angular rate about z-axis (ventral axis), if available
stimulus	Stimulus shown in the trial (e.g. image filename)
stimulus_set*	Set to which the stimulus shown belongs
correct_response	Indicates the correct movement of a response (Push, Pull or NaN; NaN given for practice trials)
device	Android device of the participant
experiment	AAT experiment name
is_practice	Boolean indicating if the current trail is a practice trial
sensor_type	Type of accelerometer sensor on the smartphone (linear or simple)

Table A.2: DataFrame columns created by function `ComputeRT`

Column Name	Description
rt	Reaction time of the participant

Table A.3: DataFrame columns created by function `FilterData()`

Column Name	Description
accelerometer_noise	Estimate of the accelerometer noise, per trial. Only present if `CalibrateAcc = True`
gyro_noise	Estimate of the gyroscope noise, per trial. Only present if `CalibrateAcc = True` and gyroscopic data is available

Table A.4: DataFrame columns created by function `ResampleData()`

Column Name	Description
`times`	Modifies `times` to be a unified time vector (i.e. consistent for both accelerometer and gyroscope) with a temporal resolution of 1 ms
`acceleration*`	Resampled acceleration along z-axis (approach-avoidance/ ventral axis)
`acceleration_x*`	Resampled acceleration along x-axis (vertical axis)
`acceleration_y*`	Resampled acceleration along y-axis (lateral axis)
`gyro_x`	Resampled angular rate about x-axis (vertical axis), if available
`gyro_y`	Resampled angular rate about y-axis (lateral axis), if available
`gyro_z`	Resampled angular rate about z-axis (ventral axis), if available

Table A.5: DataFrame columns created by function `Correct4Rotations()`

Column Name	Description
`angle_x`	Angle, about x-axis, of the phone. Only available if `StoreTheta=True`
`angle_y`	Angle, about y-axis, of the phone. Only available if `StoreTheta=True`
`angle_z`	Angle, about z-axis, of the phone. Only available if `StoreTheta=True`
`acceleration`	Rotation corrected acceleration along z-axis (approach-avoidance/ventral axis)
`acceleration_x`	Rotation corrected acceleration along x-axis (vertical axis)
`acceleration_y`	Rotation corrected acceleration along y-axis (lateral axis)

Table A.6: DataFrame columns created by function `ComputeDistance()`

Column Name	Description
`distance_x`	Displacement along x-axis
`distance_y`	Displacement along y-axis
`distance_z`	Displacement along z-axis

Table A.7: DataFrame columns created by function `ComputeDeltaAandD()`

Column Name	Description
`da`	Reaction Force per trial, if selected by user.
`dd`	Reaction Distance per trial, if selected by the user.

Table A.8: DataFrame columns created by function `FuseStimulusSets()`

Column Name	Description
`stimulus_set`	Applies "control" and "target" mapping (see section 3.6) to DataFrame

B

Appendix B: Averaged DataFrame Structure

This appendix provides some documentation on the structure of the DataFrame used to hold the averaged AAT data used for analysis. Since the rows to the DataFrame correspond to the participants, they will not be discussed further here. The main focus of this appendix is to provide an overview of the columns of the DataFrame and the data they hold. The various DataFrame columns, and their accompanying descriptions, added by `AverageWithinParticipant()` are summarized in table B.1.

For users who would like to apply the Linear-Mixed-Models analysis to the within-participant averaged data, the DataFrame outputted by `AverageWithinParticipant` needs to be restructured using `RestructureAvgDF`. The corresponding columns, and their descriptions, are presented in table B.2.

Table B.1: DataFrame columns created by function `AverageWithinParticipant()`

Column Name	Description
`PID`	mAAT assigned participant ID
`time`	Unified time array corresponding to the acceleration and distance (if selected) responses
`RT Push <Target>`	Reaction time of pushing target stimuli, averaged per participant. **Note:** <Target> is replaced by the user-specified target (e.g. 'happy')
`RT Push <Control>`	Reaction time of pushing control stimuli, averaged per participant. **Note:** <Control> is replaced by the user-specified control (e.g. 'angry')
`Acc Push <Target>`	Acceleration response of pushing target stimuli, averaged per participant. **Note:** <Target> is replaced by the user-specified target (e.g. 'happy')
`Acc Push <Control>`	Acceleration response of pushing control stimuli, averaged per participant. **Note:** <Control> is replaced by the user-specified control (e.g. 'angry')
`Dist Push <Target>`	Displacement (distance) response of pushing target stimuli, averaged per participant. **Note:** <Target> is replaced by the user-specified target (e.g. 'happy')
`Dist Push <Control>`	Displacement (distance) response of pushing control stimuli, averaged per participant. **Note:** <Control> is replaced by the user-specified control (e.g. 'angry')
`DeltaA Push <Target>`	Reaction Force of pushing target stimuli, averaged per participant. **Note:** <Target> is replaced by the user-specified target (e.g. 'happy')
`DeltaA Push <Control>`	Reaction Force of pushing control stimuli, averaged per participant. **Note:** <Control> is replaced by the user-specified control (e.g. 'angry')
`DeltaD Push <Target>`	Reaction Distance of pushing target stimuli, averaged per participant. **Note:** <Target> is replaced by the user-specified target (e.g. 'happy')
`DeltaD Push <Control>`	Reaction Distance of pushing control stimuli, averaged per participant. **Note:** <Control> is replaced by the user-specified control (e.g. 'angry')
`RT Pull <Target>`	Reaction time of pulling target stimuli, averaged per participant. **Note:** <Target> is replaced by the user-specified target (e.g. 'happy')
`RT Pull <Control>`	Reaction time of pulling control stimuli, averaged per participant. **Note:** <Control> is replaced by the user-specified control (e.g. 'angry')
`Acc Pull <Target>`	Acceleration response of pulling target stimuli, averaged per participant. **Note:** <Target> is replaced by the user-specified target (e.g. 'happy')
`Acc Pull <Control>`	Acceleration response of pulling control stimuli, averaged per participant. **Note:** <Control> is replaced by the user-specified control (e.g. 'angry')
`Dist Pull <Target>`	Displacement (distance) response of pulling target stimuli, averaged per participant. **Note:** <Target> is replaced by the user-specified target (e.g. 'happy')
`Dist Pull <Control>`	Displacement (distance) response of pulling control stimuli, averaged per participant. **Note:** <Control> is replaced by the user-specified control (e.g. 'angry')
`DeltaA Pull <Target>`	Reaction Force of pulling target stimuli, averaged per participant. **Note:** <Target> is replaced by the user-specified target (e.g. 'happy')
`DeltaA Pull <Control>`	Reaction Force of pulling control stimuli, averaged per participant. **Note:** <Control> is replaced by the user-specified control (e.g. 'angry')
`DeltaD Pull <Target>`	Reaction Distance of pulling target stimuli, averaged per participant. **Note:** <Target> is replaced by the user-specified target (e.g. 'happy')
`DeltaD Pull <Control>`	Reaction Distance of pulling control stimuli, averaged per participant. **Note:** <Control> is replaced by the user-specified control (e.g. 'angry')

Table B.2: DataFrame columns created by `RestructureAvgDF()`

Column Name	**Description**
`participant`	Index of participants (e.g. 0 corresponds to one participant, 1 to another and so on)
`correct_response`	The movement (i.e. Push or Pull)
`stimulus_set`	The control/target stimulus group
`rt`	Average reaction time for a participant, under the movement specified in `correct_response` and stimulus specified in `stimulus_set`.
`da`	Average reaction force for a participant, under the movement specified in `correct_response` and stimulus specified in `stimulus_set`.
`dd`	Average reaction distance for a participant, under the movement specified in `correct_response` and stimulus specified in `stimulus_set`.