Technische Universität Berlin
Fakultät I: Geisteswissenschaften
Institut für Sprache und Kommunikation
Fachgebiet Audiokommunikation

Technische Dokumentation zur Hörversuchs-Software WhisPER

Erstellt von André Wlodarski im Rahmen seiner Magisterarbeit am o.g. Fachgebiet.

Stand: 21.03.2014

Programmversion: WhisPER 1.8.0

Entwicklungsplattform: Matlab 7.4.0 (R2007a) / Vista 32 Bit

Inhalt

1 Anmerkungen

Dieses Dokument soll grundlegende technische Aspekte der Software WhisPER erläutern und somit Entwicklern erleichtern, an der Software zu programmieren. Innerhalb dieses Rahmens ist keine vollständige und detaillierte Dokumentation jeder einzelnen Funktion angestrebt, es sei hier auch auf die Kommentare im Quellcode hingewiesen. Ziel des Dokuments ist es, in die grundlegenden Prinzipien, die bei der Programmierung von WhisPER verfolgt wurden, einzuführen.

Für die bloße Anwendung der Software sind hier so gut wie keine Informationen zu finden. Ich möchte daher auf die **whisper-userdoc.pdf** hinweisen, die sich im Ordner **/doc** des WhisPER-Programmordners befindet. Dieser sind auch die Erklärungen für viele der verwendeten Termini zu entnehmen.

Für die Weiterentwicklung ist eine Aktualisierung der hier gesammelten Informationen erwünscht. Da dafür derzeit noch kein Modus gefunden ist, bitte ich um Kontaktaufnahme. Dies kann am Besten über die Projekt-Homepage unter

www.whisper-evaluation.de

geschehen.

WhisPER wurde von Simon Ciba und André Wlodarski im Rahmen Ihrer Magisterarbeiten am Fachgebiet Audiokommunikation der TU Berlin entwickelt.

2 Konventionen

Natürlich sind Konventionen bei der Programmierung wichtig, um einen übersichtlichen Code und einheitlichen Stil bei einem Projekt gewährleisten zu können. Auch bei WhisPER gibt es einige dieser Konventionen, die im Folgenden genannt werden.

2.1 Variablennamen

- Kleinschreibung
- Sonderzeichen werden sinnvoll ersetzt (Bsp: *ä* → *ae* oder *?* → *question_mark*)
- bei Wortgruppen / für Leerzeichen den Unterstrich als Trenner benutzen, Bsp: *stimulus_level*
- keine Abkürzungen verwenden, vor allem nicht, wenn das abgekürzte Wort nicht mehr zu erkennen wäre

Ausnahme von diesen Regeln bilden bei WhisPER die 5 globalen Struktur-Variablen **TSP, TSD, PP, PC, PS,** die in Großbuchstaben geschrieben sind und weiter unten erklärt werden.

2.2 Namen für die GUI-Programmierung

Die GUI-Elemente treten in WhisPER in besonders großer Anzahl auf. Hier ist eine konsequente Benennung im „Tag" des jeweiligen Elements nach folgenden Richtlinien gewünscht:

Push-Buttons: *pb_[Name]*

(wobei der Name der Beschriftung des Buttons entsprechen soll, dabei werden Leerzeichen durch _ ersetzt. Sollten innerhalb eines GUI-Schirms zwei oder mehr gleich beschriftete Buttons vorhanden sein, erfolgt eine zweistellige Nummerierung, beim ersten Button in Leserichtung beginnend. Bsp.: *pb_[Name]_01*)

list-Boxen: *lb_[Name]*

Menüpunkte: *men_[Name]*

Radio-Buttons: *rad_[Name]*

popup-menus: *pop_[Name]*

Edit-Text-Felder: *ed_[Name]*

Textfelder: *txt_[Name]*

(wobei Name eine sinnvoll gewählte Beschreibung bezüglich der anzunehmenden Inhalte der Felder sein soll)

uipanels: *pan_[Name]*

figures: *fig_[Name]* wobei der Name dem Dateinamen (ohne Endung) entsprechen soll

Check-Boxen: *cb_[Name]*

2.3 Layout

Die GUIs erscheinen in der Regel zentriert auf dem Bildschirm (mittels *movegui*-Aufruf in der *OpeningFcn*). Für Ausnahmen sollte es einen sinnvollen Grund geben, z.B. die Darstellung zweier GUI-Schirme zur gleichen Zeit.

Inhaltlich verwandte Elemente sollten mittels eines *uipanels* gruppiert werden, um auch optisch auf die Zusammengehörigkeit hinzuweisen.

Die Standard-Schrift in den Konfigurations-GUIs ist **MS Sans Serif** bei Größe **9**! Bitte beachten, dass MATLAB standardmäßig Schriftgröße 8 setzt.

Die Schrift für list-Boxen soll **FixedWidth** bei Größe **8** sein. Durch die normierte Breite aller Zeichen bei dieser Schriftart ist eine korrekte tabellarische Darstellung möglich.

3 Datenstruktur

Der aufwändigste Teil der Software ist die Speicherstruktur der anfallenden Daten. Da viele Routinen unabhängig voneinander auf Konfigurations- und empirische Daten zugreifen müssen, wurden global verfügbare Variablen eingeführt.

3.1 globale Variablen

Die Datenverwaltung ist in 5 globalen Struktur-Variablen organisiert, die jeweils für bestimmte Typen von Daten dienen. Es ist zu beachten, dass die Namen dieser Variablen in sämtlichen Funktionen nicht mehr vergebbar sind.

Generell sind die Daten überwiegend in Cell-Arrays, die über teilweise recht komplexe Koordinaten angesprochen werden müssen, organisiert. Dies mag auf den ersten Blick befremdlich wirken, ist aber eine gute Lösung, um Prozeduren leichter allgemeingültig programmieren zu können. Eine Organisation in flachen Strukturen wäre dafür ungeeignet.

3.1.1 TSP

TSP steht für "test series parameters". In dieser recht umfangreich gefüllten Variablen werden alle Konfigurationseinstellungen abgelegt, die die Versuchsreihe (**test series**) betreffen. Dies sind zum Einen allgemeine Daten der Versuchsreihe wie z.B. deren Name oder Daten über die zugehörigen Stimuli. Zum Anderen sind auch verfahrensspezifische Konfigurationsdaten enthalten. Die verschiedenen Verfahren benutzen dabei innerhalb der ihnen zugewiesenen Datenbereiche unterschiedliche Strukturen, die von der Beschaffenheit des Verfahrens abhängen.

Hinweis: NICHT enthalten sind hier Daten, die die allgemeine Konfiguration des Programms betreffen. (also z.B. die Plotting-Einstellungen)

Die **TSP** wird auch als **TSP.mat** gespeichert und im Versuchsreihen-Ordner mitgeführt.

Wegen der komplexen Struktur befindet sich eine Übersicht über den Aufbau im Anhang dieses Dokuments. Diese ist sehr hilfreich bei der Arbeit mit der **TSP**.

3.1.2 PC

Die globale Variable **PC** (‚program components') hält Strukturen bereits, die Teil der Software sind. Dies sind Leer-und Initialisierungs-Datenblöcke für die einzelnen Verfahren. Die Strukturen aus der **PC** werden beim Einfügen eines Verfahrens an die entsprechende Stelle in der **TSP** übertragen.

Die **PC.mat** wird mit der Software im Programm-Ordner mitgeliefert und bei Start der Software in den Arbeitsspeicher geladen.

3.1.3 TSD

TSD steht für „test series data“. Diese Variable beinhaltet sämtliche empirische Daten. Die Struktur der **TSD** ist so angelegt, dass auch zukünftig zu implementierende Verfahren den bestehenden Programmkern nutzen können. Die Idee ist, dass jede **test section** ein Feld auf oberster Ebene mit einem Cell-Array belegt. Dieses bietet natürlich wieder beliebig viele Spalten, um jede benötigte Anzahl an Variablen für den späteren Export aufnehmen zu können. Ziel für den Datenexport ist schließlich immer eine spaltenweise Darstellung der erfassten Variablen (im statistischen Sinne), um diese in Programme wie **SPSS** oder **Excel** einfach importieren zu können.

In der ersten Zeile der **TSD** ist Platz für die entsprechenden Überschriften, die von den jeweiligen Verfahrensroutinen angelegt werden. So kann für jedes Verfahren eine andere Konfiguration der Spaltenbelegung realisiert werden.

Die **TSD** wird auch als **TSD.mat** im Versuchsreihen-Ordner gespeichert.

3.1.4 PP

Die globale Variable **PP** (‚program parameters’) führt Informationen, die WhisPER zur Laufzeit benötigt. Dies sind z.B. der Speicherort der aktuell geladenen Versuchsreihe oder Daten über die Zusammenstellung der Hörbeispiele eines Versuchs-**Run**s.

Die Struktur der **PP** ist flach und kann schnell durch Betrachtung im MATLAB-Array-Editor erfasst werden.

Die **PP** enthält keine zur Speicherung relevanten Daten und existiert daher nur zur Laufzeit im Arbeitsspeicher.

3.1.5 PS

In der **PS** (‚program settings‘) werden Programmeinstellungen von WhisPER gespeichert. Derzeit sind das die Netzwerk- und die Plotting-Einstellungen.

Die **PS** wird als **PS.mat** im Programm-Ordner gespeichert. Es handelt sich also um eine lokale Speicherung, die dem Computer zugeordnet ist, der WhisPER ausführt, nicht jedoch der Versuchsreihe.

4 Programm- und Quellcodestruktur

Nun soll erklärt werden, wie die Software grundsätzlich arbeitet und wie der Quellcode organisiert ist.

4.1 Funktionale Programmierung

WhisPER ist, bis auf wenige Ausnahmen, funktional programmiert. Die Funktionen befinden sich überwiegend in den .m-Dateien, die vom GUI-Editor *GUIDE* beim Entwurf der GUIs vorgeneriert wurden. Der Vorteil dabei ist, dass Aktionen die auf Knopfdruck im GUI-System hervorgerufen werden, bei der Programmierung direkt über den View-Callback-Link in *GUIDE* aufgerufen und editiert werden können. Dies ist also vor allem eine praktische Erleichterung. Außerdem gestaltet sich der Zugriff auf die GUI-Handles so recht einfach.

Auch weitere Funktionen, die nicht direkt mit einem GUI-Element in Verbindung stehen, sondern an vielen verschiedenen Stellen dieser GUI-Instanz aufgerufen werden müssen, befinden sich in der Regel in der gleichen m-Datei, um die Gesamt-Anzahl an Dateien, die für WhisPER benötigt werden, geringer zu halten.

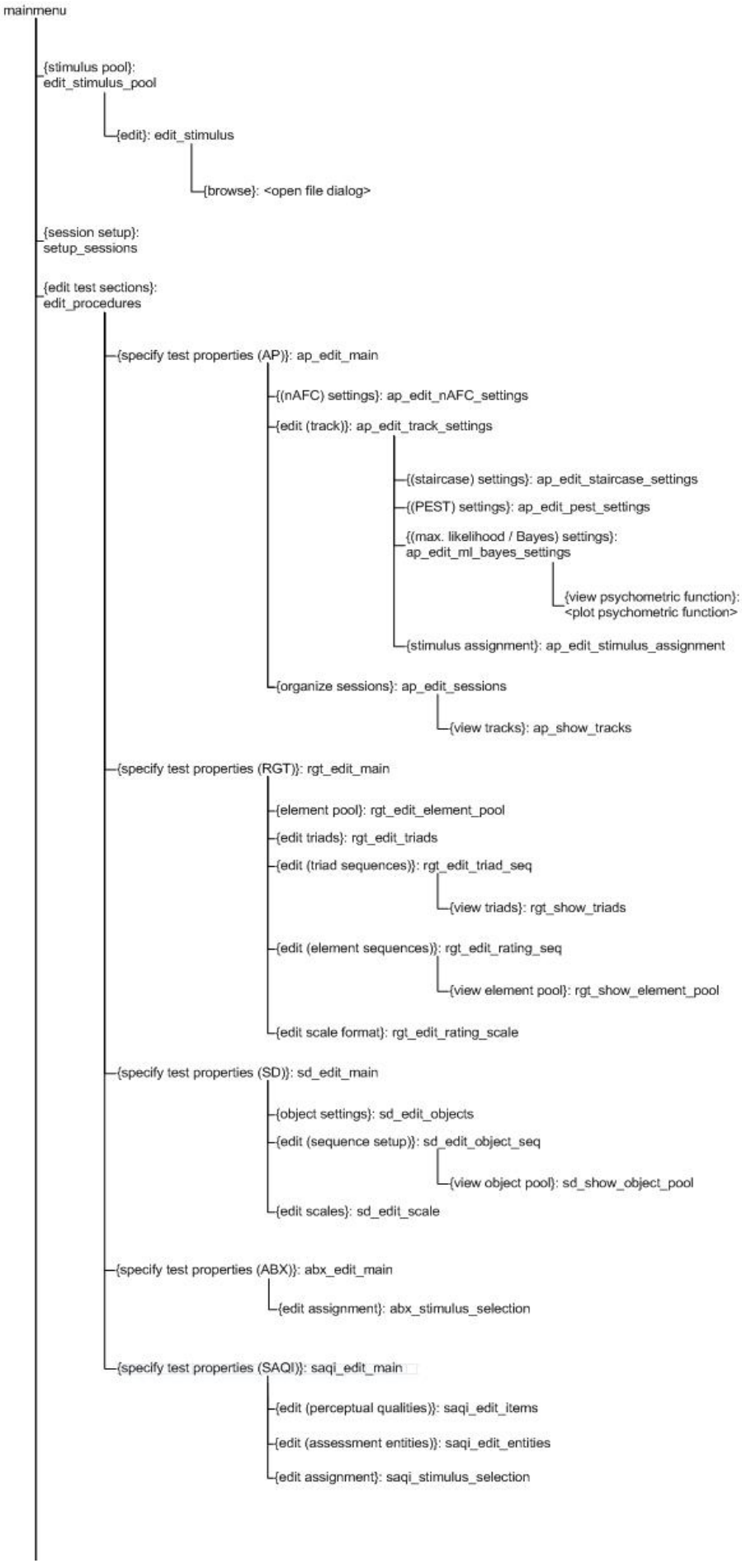
Lediglich Funktionen, die aus verschiedenen GUIs aufgerufen werden, sind in separaten .m-Dateien abgelegt (Bsp: *str_to_len.m*).

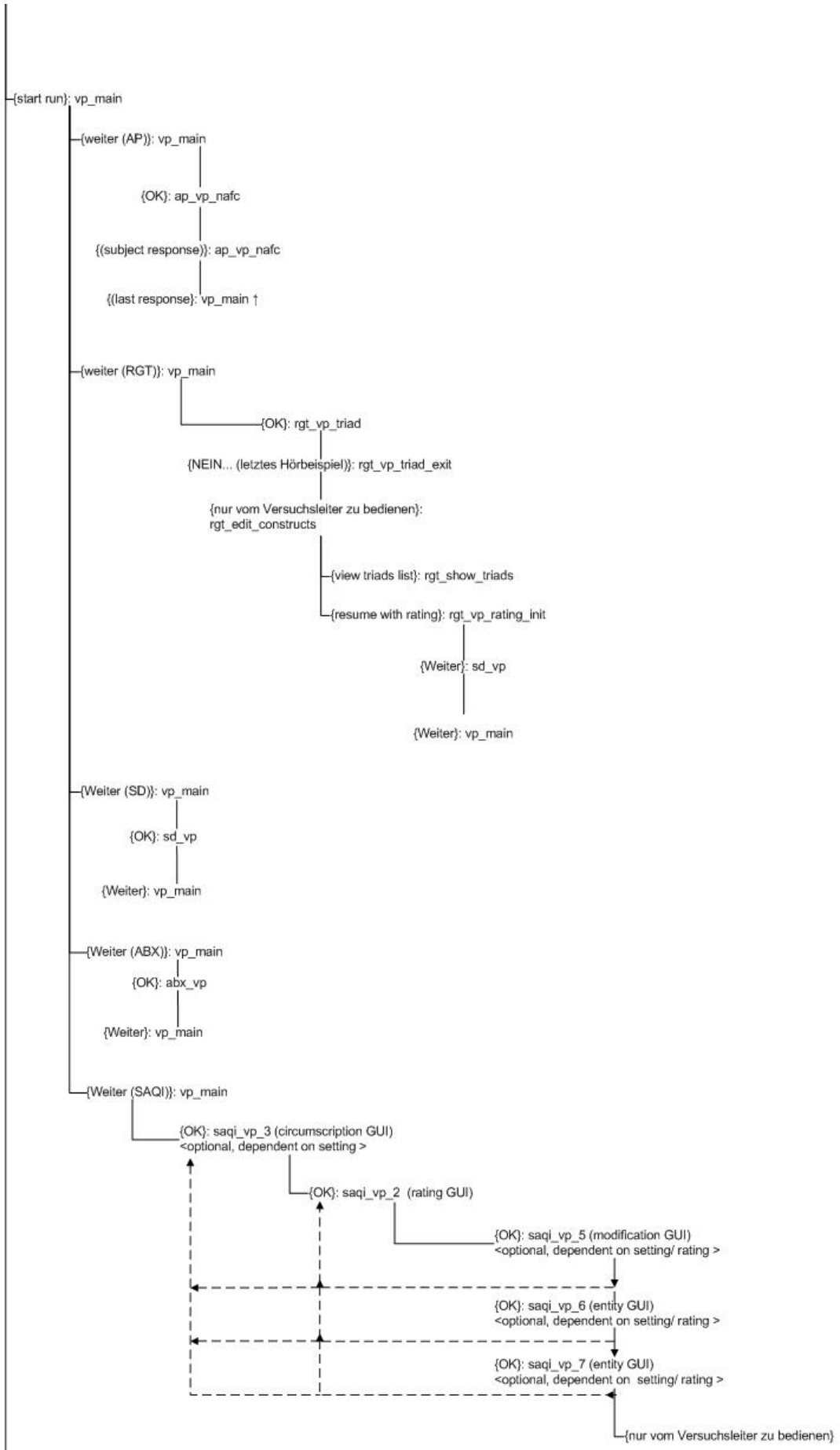
4.2 Verknüpfungen und Struktur der Benutzeroberfläche

WhisPER verfügt über viele verschiedene GUI-Bildschirme. Im Folgenden wird der Einfachheit halber nur der Begriff „GUI“ verwendet, der jedoch nicht für die Gesamtheit der Benutzeroberfläche, sondern für einzelne GUI-Schirme stehen soll.

Um einen Überblick darüber zu gewinnen, wann und welche Aktionen des Benutzers auf einen anderen Schirm führen, und welche GUIs überhaupt von anderen zu erreichen sind, wurde zusammen mit Simon Ciba ein Struktur -Diagramm entworfen. Dieses beginnt mit dem Eintrag *mainmenu*. In der Endversion der Software ist dem noch die Datei *whisper.m* vorgeschaltet, welche den Startpunkt für den Benutzer darstellt und nichts weiter tut, als den Workspace zu leeren und die *mainmenu.m* aufzurufen. Diese Datei benutzt daher auch keine Oberfläche.

Zum besseren Verständnis der Grafik sei auf die Legende am Ende (der 3seitigen Grafik) hingewiesen.


mainmenu
{stimulus pool}:
edit_stimulus_pool
{edit}: edit_stimulus
{browse}: <open file dialog>
{session setup}:
setup_sessions
{edit test sections}:
edit_procedures
{specify test properties (AP)}: ap_edit_main
{(nAFC) settings}: ap_edit_nAFC_settings
{edit (track)}: ap_edit_track_settings
{(staircase) settings}: ap_edit_staircase_settings
{(PEST) settings}: ap_edit_pest_settings
{(max. likelihood / Bayes) settings}:
ap_edit_ml_bayes_settings
{view psychometric function}:
<plot psychometric function>
{stimulus assignment}: ap_edit_stimulus_assignment
{organize sessions}: ap_edit_sessions
{view tracks}: ap_show_tracks
{specify test properties (RGT)}: rgt_edit_main
{element pool}: rgt_edit_element_pool
{edit triads}: rgt_edit_triads
{edit (triad sequences)}: rgt_edit_triad_seq
{view triads}: rgt_show_triads
{edit (element sequences)}: rgt_edit_rating_seq
{view element pool}: rgt_show_element_pool
{edit scale format}: rgt_edit_rating_scale
{specify test properties (SD)}: sd_edit_main
{object settings}: sd_edit_objects
{edit (sequence setup)}: sd_edit_object_seq
{view object pool}: sd_show_object_pool
{edit scales}: sd_edit_scale
{specify test properties (ABX)}: abx_edit_main
{edit assignment}: abx_stimulus_selection
{specify test properties (SAQI)}: saqi_edit_main
{edit (perceptual qualities)}: saqi_edit_items
{edit (assessment entities)}: saqi_edit_entities
{edit assignment}: saqi_stimulus_selection


{start run}: vp_main
{weiter (AP)}: vp_main
{OK}: ap_vp_nafc
{{subject response}}: ap_vp_nafc
{{last response}: vp_main ↑
{weiter (RGT)}: vp_main
{OK}: rgt_vp_triad
{NEIN... (letztes Hörbeispiel)}: rgt_vp_triad_exit
{nur vom Versuchsleiter zu bedienen}:
rgt_edit_constructs
{view triads list}: rgt_show_triads
{resume with rating}: rgt_vp_rating_init
{Weiter}: sd_vp
{Weiter}: vp_main
{Weiter (SD)}: vp_main
{OK}: sd_vp
{Weiter}: vp_main
{Weiter (ABX)}: vp_main
{OK}: abx_vp
{Weiter}: vp_main
{Weiter (SAQI)}: vp_main
{OK}: saqi_vp_3 (circumscription GUI)
<optional, dependent on setting >
{OK}: saqi_vp_2 (rating GUI)
{OK}: saqi_vp_5 (modification GUI)
<optional, dependent on setting/ rating >
{OK}: saqi_vp_6 (entity GUI)
<optional, dependent on setting/ rating >
{OK}: saqi_vp_7 (entity GUI)
<optional, dependent on setting/ rating >
{nur vom Versuchsleiter zu bedienen}

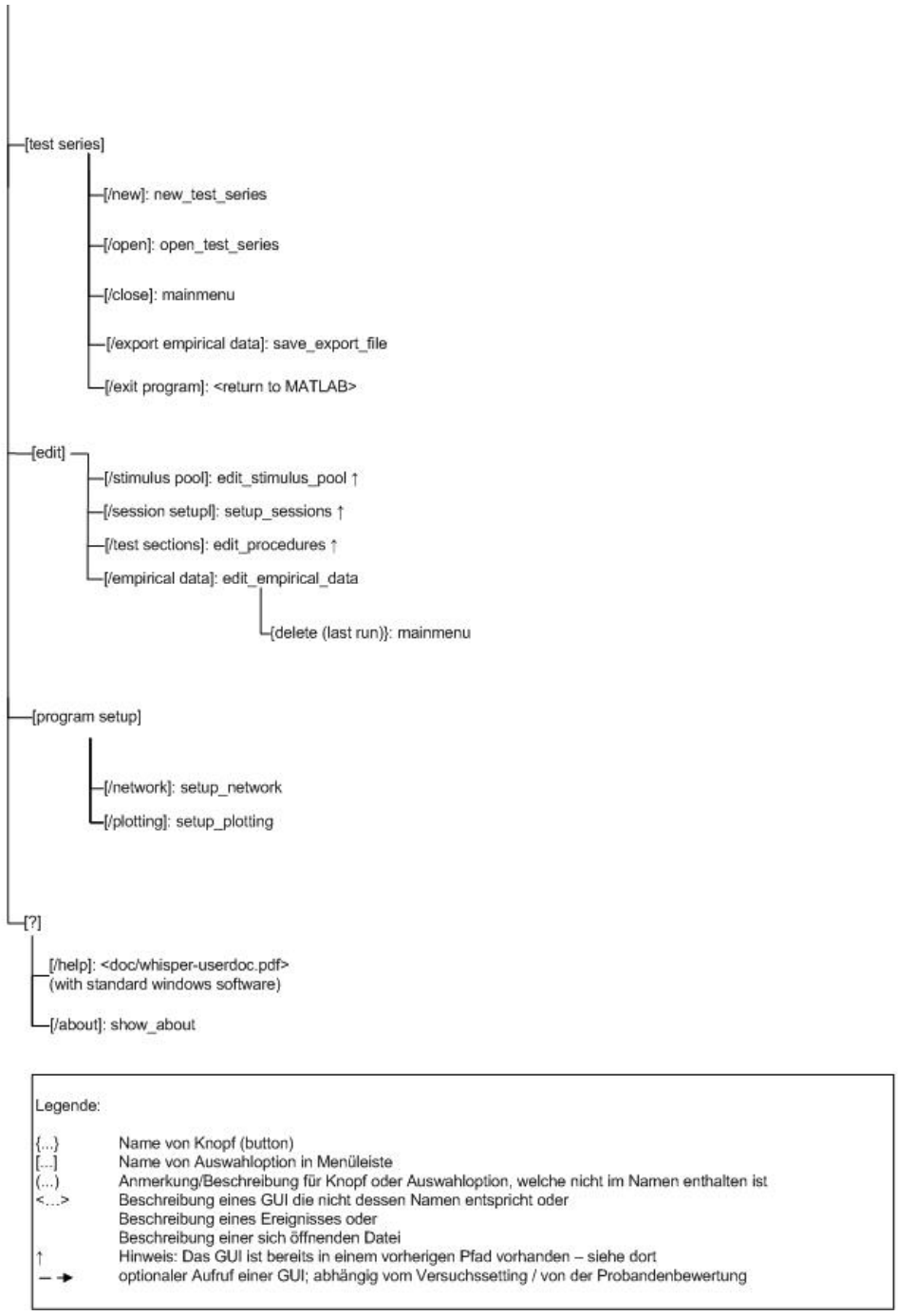


Abbildung 1 – GUI-Struktur

4.3 Programminterne Kommunikation

Die Übergabe von GUI zu GUI soll nicht über einen Funktionsaufruf der Form *gui2('Daten')* erfolgen, da Inkompatibilität mit *GUIDE*-GUIs neuerer MATLAB-Versionen bestehen wird. Es wird daher für derartige Anwendungen die globale Variable **PP** verwendet. Dies ist kein besonders eleganter Programmier-Stil, war jedoch leider nötig, da man bei MATLAB diesbezüglich einigen Beschränkungen unterlegen ist.

Möglicherweise werden zukünftige Versionen von *GUIDE* dahingehend verbessert. In dem Fall wäre eine Umstellung auf die derzeit nicht verwirklichte Variante wünschenswert.

4.4 Weitere Backend-Funktionen

Hier sollen wichtige Funktionen des Backends und deren Wirkungsweise vorgestellt werden.

4.4.1 str_to_len

Diese Funktion dient der Kürzung oder Verlängerung von Strings auf eine anzugebende Länge. Sie löscht entweder die letzten Zeichen eines „zu langen“ Strings oder fügt Leerzeichen zu einem „zu kurzen“ String hinzu und ist somit ein kleiner Helfer bei der Formatierung von Daten für die tabellarische Darstellung in list-Boxen.

4.4.2 load_test_series

Diese Funktion wird von der GUI *open_test_series* aufgerufen. Sie hat die Aufgabe, alle relevanten vorhandenen Variablen zu löschen und die Daten der neu zu öffnenden Versuchsreihe in den Speicher zu laden.

Eine Besonderheit ist, dass hier beim Laden der Versuchsreihe diese in der Datenstruktur automatisch umbenannt wird, falls zuvor der Name des Versuchsreihen-Ordners geändert wurde. So wird gewährleistet, dass Versuchsreihen- und Ordnername jederzeit übereinstimmen.

4.4.3 present_stimulus

Diese Funktion spielt den Stimulus ab, dessen **Stimulus-ID** als Parameter übergeben wurde und versendet dem Stimulus zugeordnete OSC-Befehle.

Für die Wiedergabe von Wave-Dateien wird derzeit ausschließlich der *audioplayer* von MATLAB genutzt, es werden 1- und 2kanalige Dateien unterstützt. Für zukünftige Versionen wäre eine ASIO-Implementierung wünschenswert. Diese kam bisher nicht zustande, da die für diesen Zweck angedachte Bibliothek pa-wavplay keine Möglichkeit bietet, einmal gestartete Wiedergaben vor dem Dateiende zu beenden. *Anmerkung: Möglicherweise wäre durch eine geschickte Neukompilierung der portaudio-Bibliothek (die mittlerweile in der Version 2 vorliegt) für MATLAB unter Windows das Problem zu*

lösen. Dies wäre eine Aufgabe für einen Programmierer, der gute Kenntnisse in den dafür benötigten Sprachen hat.

Je Stimulus können bis automatisch bis zu sechs unabhängige OSC-Befehle (siehe opensoundcontrol.org) ausgelöst werden. Seit der whisPER version 1.8.0 wird dazu **nicht mehr** die von Andy Schmeder erstellte und auf *.dlls basierende freie Matlabimplementation für OSC verwendet. Stattdessen wurde eine Matlab-native Variante gesucht. Ursache dafür war die auslaufende Unterstützung von *.dlls durch Matlab sowie eine vereinfachte Kompatibilität der OSC-Implementierung über verschieden Betriebssysteme hinweg.

Auf Matlab® Central konnte dafür eine auf der Instrument Control Toolbox (im Umfang der Studentenversion von Matlab enthalten) von Matlab basierende Implementierung von Mark Marijnssen gefunden werden[1]. Diese nutzt die durch die Toolbox bereitgestellten UDP-Fähigkeiten Matlabs aus. **Diese Anpassung bedeutet aber auch, dass whisPER zum korrekten Funktionieren künftig die Instrument Control Toolbox benötigt.**

4.4.4 save_export_file

Die Funktion *save_export_file* speichert die Datenexport-Datei für die als Parameter übergebene **test section** im Versuchsreihen-Ordner.

Dazu werden die verschachtelten Ebenen der **TSD** ausgelesen und in ein zweidimensionales Array übernommen, welches dann gespeichert werden kann. Es müssen einige verschachtelte Schleifen und Bedingungen benutzt werden um das zu realisieren. Sie sind in der Datei entsprechend kommentiert.

Der große Gewinn dieser zuerst etwas kompliziert wirkenden Methode ist, dass sie für jede Art von Testverfahren, das zukünftig implementiert wird, funktioniert, so lange die Datenstruktur der **TSD** (siehe Anhang) eingehalten wird. Dies sollte für jedes vorstellbare Testverfahren realisierbar sein. Darüberhinaus wird ein Export-File generiert, das stets nur so viele Spalten enthält, wie auch Datenvariablen vorhanden sind. Es gibt also beim Import in Auswertungs-Software keine leeren Spalten, die erst entfernt werden müssen.

Exportiert werden die Daten schließlich als eine durch Semikolon getrennte csv-Datei je Versuchsteil (**test section**). Dies geschieht automatisch nach jedem **run**, aber auf Wunsch auch manuell aus dem Hauptmenü.

Für SD- und RGT-sections werden zusätzlich die Formate GRD (Idiogrid) und GridXML (GridSuite) bei Aufruf aus dem Hauptmenü in das export-Subverzeichnis geschrieben. (Details s. User-Doc „subfolder export“). Die zugehörige Funktion ist save_GridExport

[1] http://www.mathworks.fr/matlabcentral/fileexchange/31400-send-open-sound-control-osc-messages/content/oscsend.m

mit den beiden Basis-Funktionen save_export_whisper2idiogrid und save_export_whisper2gridXML.

4.4.5 save_testseries_info

Diese Funktion speichert Informationen über die Konfigurationsdaten der Versuchsreihe in die Datei **testseries.info** im Versuchsreihen-Ordner.

Diese info-Datei dient vor allem dem Versuchsleiter, da alle konfigurierten Parameter der Versuchsreihe mit ihrer Hilfe nachvollzogen werden können. Sie wird bei jedem Aufruf des mainmenu neu geschrieben, also beispielsweise nach dem Verlassen des Stimulus-Pool oder vor dem Start eines Runs. Sie wird somit immer sehr aktuell gehalten.

Gleichzeitig dient die Datei aber auch dazu, den Versuchsreihen-Ordner beim Öffnen einer Versuchsreihe als gültigen Ordner zu identifizieren. Konkret wird bei Auswahl eines Ordners im **open**-Dialog ihr Vorhandensein überprüft.

4.4.6 Adaptions- und Schätzalgorithmen

Die Funktionen ***MLBayes***, ***mvPEST***, ***PEST***, ***staircase1d1u***, ***staircase2d1u*** und ***staircase3d1u*** sind die Implementierungen der Adaptionsmechanismen und Schätzmethoden welche für die Durchführung adaptiver Verfahren benötigt werden. Die Funktionen ***plot_prior*** und ***plot_psyfunc*** ermöglichen die grafische Darstellung bestimmter Verfahrensparameter innerhalb der Benutzeroberfläche im Zuge der Konfiguration der Verfahren. Die genannten Funktionen wurden von Simon Ciba erstellt und werden im Anhang 2 dieses Dokuments eingehender beschrieben.

4.4.7 Funktionen externer Autoren

Zum Teil wurden Funktionen externer Autoren verwendet. Diese sollen hier kurz genannt werden.

setfigdocked dient der Darstellung des Basis-Schirms. MATLAB bietet keine komfortable Möglichkeit an, Fenster per Befehl automatisch zu maximieren. Mittels dieser Funktion konnte dies einfach realisiert werden (Informationen dazu unter http://www.mathworks.com/matlabcentral/fileexchange/18106)

Für whisPER Versionen seit v1.8.0:

WhisPERs OSC-Fähigkeiten werden nun von Mark Marijnssens[2] auf der Instrument Control Toolbox (im Umfang der Studentenversion von Matlab enthalten) von Matlab basierenden ***oscsend.m*** realisiert. **Diese Anpassung bedeutet aber auch, dass**

[2] http://www.mathworks.fr/matlabcentral/fileexchange/31400-send-open-sound-control-osc-messages/content/oscsend.m

whisPER zum korrekten Funktionieren künftig die Instrument Control Toolbox benötigt.

Implementationsbeispiel:

In **vp_main.m** wird eine generische OSC-Stop-Botschaft initialisiert:

```
%Create OSC-stop-message
PP.osc_stop_path = '/stop';
PP.osc_stop_type = 'i';
PP.osc_stop_data = 1;
```

Diese kann nun von jeder Stelle des whisPER-Codes auf einfache Weise ausgelöst werden:

% Send OSC-stop-message via every pre-configured OSC-server: insert this code block at the % desired position into the m-code of your listening test module

```
global PP
for n = 1:numel(PP.u)
	fopen(PP.u{n});
	oscsend(PP.u{n},PP.osc_stop_path,PP.osc_stop_type,PP.osc_stop_data);
	fclose(PP.u{n});
end
```

Für whisPER Versionen vor v1.8.0:

Die Bibliotheken ***osc_free_address.dll, osc_free_server.dll, osc_new_address.dll, osc_new_server.dll, osc_recv.dll, osc_send.dll*** stammen aus der MATLAB-OSC-Implementierung (mehr auf opensoundcontrol.org), eine von Andy Schmeder erstellte Windows-Kompilierung der liblo-Bibliotheken, die unter der GPL verfügbar sind.

5 Hinzufügen neuer Verfahren

Zuletzt sollen einige Hinweise zur Vorgehensweise bei der Implementierung neuer Verfahren unter WhisPER gegeben werden. Es gibt einige Stellen in der Software, an denen ein Verfahren integriert werden muss.

Damit das Kapitel eine prägnante Kürze behält, werden einige der bereits für die vorhandenen Verfahren implementierten kleinen Schritte hier nicht gesondert erklärt. Diese lassen sich meist einfach übernehmen.

5.1 Eintrag des Testverfahrens in der PC

Unter **PC.testing_procedures** haben alle in der Software verfügbaren Testverfahren einen Eintrag, der sich über eine Zeile erstreckt. Ihnen wird dort eine ID in der Form ‚tid[Zahl]' (begrifflich als Abkürzung für Type-ID) zugeordnet, sowie ein Name.

*Hinweis: Mit dem MATLAB-Array-Editor lässt sich dies anhand der vorhandenen Verfahren gut nachvollziehen, dazu bitte einfach die **PC** mittels Befehl „global PC" im Workspace verfügbar machen und betrachten.*

Diese beiden Felder sind zwingend auszufüllen, um die Verfahren als verfügbare Testverfahren in der GUI *edit_procedures* anzeigbar zu machen.

5.1.1 Initialisierungsdaten in der PC

Wenn ein bestimmtes Testverfahren angewendet werden soll, muss dies in der Regel immer vom Versuchsleiter in irgendeiner Form konfiguriert werden. Diese Konfigurationsdaten müssen zwar später in der **TSP** gespeichert werden, es bietet sich jedoch an, einen initialen Datensatz anzulegen. In diesem sollte die Struktur der Konfiguration, unabhängig davon, wie sie bei dem Verfahren genau beschaffen ist, mit sinnvoll voreingestellten Werten abgelegt sein und in der **PC** gespeichert werden. Dazu kann die dritte Spalte genutzt werden.

5.2 Konfigurationsdaten in der TSP

Der zuvor beschriebene Datensatz muss beim Hinzufügen des Verfahrens durch den Benutzer/Versuchsleiter (GUI *edit_procedures*) in die **TSP** kopiert werden. In der Regel sollte dieser Datensatz aus einem einzigen Cell-Array bestehen, welches natürlich beliebig viele untergeordnete Strukturen enthalten kann. Dieses wird in die Spalte 4 der **TSP** kopiert. Die anderen Spalten werden mit der ID des Verfahrens sowie internem und öffentlichem Namen des Versuchsabschnitts gefüllt.

Im Fall des RGT ist abweichend bereits eine Variante verwirklicht, die zwei Cell-Arrays benutzt. So ist die Behandlung von Semantischem Differential und dem RGT-Rating für viele Prozeduren identisch.

Natürlich müssen nun neue GUIs zur Konfiguration des Verfahrens bereit stehen, die mit dem Button „edit procedure properties (of selected test section)“ verknüpft werden und eine adäquate Anpassung der Konfigurationsdaten ermöglichen.

Konfigurations-GUIs nutzen die Variable **PP.edit_section_line** als Index für die zu bearbeitende Zeile in der **TSP**.

5.3 Start des Verfahrensablaufs in der vp_main.m

Nun muss das Verfahren in die Ablaufsteuerung integriert werden. Folgende Erläuterungen betreffen daher die Ebene, die dem Versuchsteilnehmer zugängig ist.

5.3.1 Funktion pb_weiter_Callback

Hier wird lediglich die initiale Instruktion für das nächstfolgende Verfahren eingeblendet. Der anzuzeigende Text sollte im Allgemeinen innerhalb der Konfigurationsdaten an einer nicht fest definierten Stelle bereit liegen. Daher ist hier ein entsprechender Eintrag für das neue Verfahren nötig.

5.3.2 Funktion pb_ok_Callback

Innerhalb der Funktion *pb_ok_Callback* werden die Parameter für den aktuellen **run** (Nummer des Durchlaufs liegt in **PP.run_number**) der aktuellen **section** (section-Nummer liegt in **PP.run_section_line**) festgelegt bzw. generiert. Diese sind, da sie nur für den Versuchsablauf gelten, als temporäre Daten anzusehen. Zu Ihrer Speicherung sollte daher die **PP** verwendet werden. Eine genaue Erklärung, wie diese Daten aussehen, kann hier nicht erfolgen, da dies je nach Art des Verfahrens und des dazugehörigen **session**-Nutzungskonzepts sehr unterschiedlich sein kann.

Es sollen zumindest einige Beispiele genannt werden. Bei den bereits implementierten Verfahren wird unter Anderem immer das Array **PP.playlist** genutzt. Dies enthält bei den adaptiven Verfahren die abzufragenden Tracks für den aktuellen **run**. Beim Semantischen Differential hingegen wird **PP.playlist** genutzt, um die Objekte aufzunehmen. Hier geschieht auch die Randomisierung der Abspielreihenfolge. Einige Verfahren nutzen auch die Zählvariable **PP.counter**.

5.4 Daten in der TSD speichern

Das neu zu implementierende Verfahren muss während oder am Ende seines Ablaufs die empirischen Daten speichern. In welcher Form diese in der **TSD** angelegt werden müssen, ist den Plänen im Anhang 1 zu entnehmen. Die Einhaltung der Struktur ist besonders wichtig, damit die Daten später ordnungsgemäß in eine csv-Datei übertragen werden können.

5.5 testseries.info und Protokollierung

Für jedes Testverfahren werden die vom Versuchsleiter konfigurierten Daten in der testseries.info gespeichert. Für neu hinzukommende Verfahren sollte dies ebenfalls geschehen, daher sind die entsprechenden Routinen in der Funktion *save_testseries_info* zu ergänzen. Die Funktion ist selbsterklärend.

Je nachdem, welcher Art das Verfahren ist, können in seinen Ablauf Protokollierungen integriert werden. Dazu muss die Funktion *write_to_log* an den entsprechenden Stellen eingebracht werden.

Anhang 1 der Technischen Dokumentation – Datenstruktur der Variablen TSP und TSD

Siehe Excel Tabelle *Datenstruktur der Variablen TSP und TSD*

Anhang 2 der Technischen Dokumentation – Adaptions- und Schätzalgorithmen

Das folgende Dokument und die darin beschriebenen Funktionen wurden von Simon Ciba im Rahmen seiner Magisterarbeit erstellt.

Beschreibungen der Adaptions- und Schätzalgorithmen + Plot-Funktionen (Teil des Anhangs der technischen Dokumentation zu WhisPER; S. Ciba/16.11.2008)

Hinweise:

- Die Abfolgen der einzelnen im Folgenden dargestellten Parameter entsprechen den Reihenfolgen, in welchen diese übergeben werden.

- Die einzelnen Parameter sollen hier jeweils in ihrer Bedeutung nur kurz angerissen werden. Teilweise werden sie auch nur genannt. Für eine ausführlichere Erklärung dieser sei auf die jeweils angegebenen Publikationen verwiesen. Außerdem werden die implementierten adaptiven Verfahren in meiner Magisterarbeit dargestellt.

I. Adaptions- und Schätzalgorithmen

staircaseXd1u.m (X=1, 2, 3) (Cornsweet, 1968; Levitt, 1971)

Mechanismus eines einfachen bzw. transformierten, einer X-Down/1-Up-Regel (X=1, 2, 3) folgenden Staircase-Verfahrens; Halbierungen der Schrittweite bei vorgegebenen Reversals; Schätzung der Schwelle durch Mittelung der Reizstärken bei einer vorgegebenen Anzahl letzter aufeinander folgender Reversals (vgl. *Mid-Run-Estimates*).

Ausgabeparameter		
x_next	Integer	Reizstärkenwert für die Darbietung im nächsten Trial, einer X-Down/1-Up-Regel folgend.
x_est	Double	Schätzung der Schwelle und gleichzeitig Indikator für Terminierung (x_est = 0: das Verfahren wurde noch nicht terminiert, d.h. es liegt noch keine Schätzung vor).
Eingabeparameter		
x_hist	Integer-Array	Enthält die Reizstärkenwerte sämtlicher bisher dargebotener Stimuli.
r_hist	Integer-Array	Enthält die zugehörigen Antworten (+1: positiv, -1: negativ).
range	Integer	Legt den Bereich [1, *range*] möglicher Reizdarbietungen (Range) fest, welcher in Schritten von 1 aufgelöst wird.
nrmax	Integer	Maximale Anzahl an Umkehrungen (Reversals) des Tracks. Bei Erreichen dieser Zahl wird das Verfahren terminiert.
halvings	Integer-Array	Enthält die Indizes der Umkehrpunkte, bei denen Halbierungen der Schrittweite erfolgen sollen.
numlastr	Integer	Anzahl der letzten aufeinander folgenden Reversals, welche in die Schätzung der Schwelle eingehen sollen.

PEST.m (Taylor und Creelman, 1967) und mvPEST.m (Findlay, 1978)

Mechanismus nach dem PEST-Verfahren - einmal in der Originalfassung (PEST.m) und außerdem in der modifizierten Version des sogenannten „More-Virulent-PEST“ (mvPEST.m).

Ausgabeparameter

x_next	Integer/ Double	Reizstärkenwert für die Darbietung im nächsten Trial. (Nach dem letzten Funktionsaufruf ist dieser Wert gleich der Schätzung der Schwelle in x_est.)
x_est	Double	Schätzung der Schwelle und gleichzeitig Indikator für Terminierung (x_est = 0: das Verfahren wurde noch nicht terminiert, d.h. es liegt noch keine Schätzung vor).

Eingabeparameter

x_hist	Integer-Array	Enthält die Reizstärkenwerte sämtlicher bisher dargebotener Stimuli.
r_hist	Integer-Array	Enthält die zugehörigen Antworten (+1: positiv, -1: negativ).
range	Integer	Legt den Bereich [1, *range*] möglicher Reizdarbietungen (Range) fest, welcher in Schritten von 1 aufgelöst wird.
dx_ini	Integer	Anfangswert der Schrittweite (Zweierpotenz).
dx_max	Integer	Maximale Schrittweite, welche während des Verfahrens zu keinem Zeitpunkt (auch nicht im ersten Schritt) überschritten wird (Zweierpotenz).
P_t	Double	Konvergenzniveau des Tracks (aus [0, 1]).
Nur für PEST.m		
W	Double	Deviation-Limit für Signifikanztest. *
Nur für mvPEST.m		
M	Integer	Parameter M. *

MLBayes.m (Lieberman & Pentland, 1982; * Watson & Pelli, 1983; † King-Smith et al., 1994; ‡ Treutwein, 1997)

Mechanismus, welcher die Merkmale verschiedener Maximum-Likelihood- bzw. Bayes-Verfahren in sich vereint und diese über bestimmte Eingabeparameter zur Auswahl stellt. Der grundlegende Kern[3] des Algorithmus basiert auf der Implementierung des Best-PESTVerfahrens nach Lieberman und Pentland (1982). Diejenigen Parameterwerte, welche aus Erweiterungen resultieren, die anderen Verfahren entnommen wurden, sind mit Verweis auf die entsprechenden Publikationen gekennzeichnet.

Ausgabeparameter

x_next	Integer	Reizstärkenwert für die Darbietung im nächsten Trial (beinhaltet A-priori-Information, wenn diese spezifiziert ist).
x_est	Integer/ Double	Abschließende Schätzung der Schwelle (hiervon kann auf Wunsch die A-priori-Information ausgeschlossen werden, s. Parameter *exclude*). x_est = 0: das Verfahren wurde noch nicht terminiert. Wenn *averagetype==2* gewählt ist, kann der Parameter auch reelle Werte aus [1, *range*] enthalten.
finalLike	Double-Array	Enthält die Likelihood-Funktion der abschließenden Schätzung.

[3] Um den „Kern“ des Algorithmus aus dem Quellcode herauszulesen, bedarf es eines sehr geduldigen Blickes. Es sei an dieser Stelle auch auf eine weitere Veränderung gegenüber dem Original hingewiesen: anstelle der Logarithmierung der Likelihood-Funktion wird eine Normierung auf die Fläche unter dem Funktionsgrafen durchgeführt. Die normierte ikelihood wird innerhalb der Bayes’schen Theorie auch als A-posteriori-Wahrscheinlichkeitsdichte bezeichnet (Treutwein 1995, 1997).

Eingabeparameter

x_hist	Integer-Array	Enthält die Reizstärkenwerte sämtlicher bisher dargebotener Stimuli.
r_hist	Integer-Array	Enthält die zugehörigen Antworten (+1: positiv, -1: negativ).
range	Integer	Legt den Bereich [1, *range*] möglicher Reizdarbietungen (Range) fest, welcher in Schritten von 1 aufgelöst wird.
beta	Double	Spread-Parameter.
lambda	Double	Lapsing-Rate (aus [0, 1-*gamma*]).
n_or_gamma	Integer/Double	Festlegung des Antwortparadigmas (Werte größer oder gleich 2: Anzahl n der Intervalle beim nAFC-Paradigma, Werte aus [0, 1]: False-Alarm-Rate gamma für ja/nein-Paradigma).
Priortype	Integer	Gibt an, auf welche Art A-priori-Information berücksichtigt werden soll. (1: implizite Trials nach Lieberman & Pentland, 1982, 2: gaussförmige A-priori-Likelihood (*), sonst: "non-informative prior", d.h. konstante A-priori-L. mit erster Reizdarbietung in der Mitte des Range).
mu	Double	Mittelwert der gaussförmigen a-priori-Likelihood. (*)
sigma	Double	Standardabweichung der gaussförmigen A-priori-Likelihood. (*)
averagetype	Integer	Gibt an, nach welcher Methode die Mittelwertbildung erfolgen soll (1: Modalwert, d.h. Maximum-Likelihood-Methode, 2: arithmetischer Mittelwert). (†, ‡).
exclude	Integer	Gibt an, ob A-priori-Information von der Berechnung der abschließenden Schätzung ausgeschlossen werden soll (1: Ausschluss und sonstige: kein Ausschluss). (*).
termtype	Integer	Gibt an, auf welche Weise das Verfahren terminiert werden soll (0: Terminierung per vorgegebenem Konfidenzintervall (‡), ganzzahlige Werte größer als 1: Terminierung nach einer fest vorgegebenen Anzahl an Trials; letztere entspricht dann gleichzeitig dem Parameterwert).
ConCoeff	Double	Konfidenzkoeffizient (Werte aus [0, 1]) (‡).
CI_max	Double	Obere Schranke für das Konfidenzintervall, bei deren Erreichen bzw. Unterschreiten das Verfahren terminiert wird. Werte müssen positiv und sollten sinnvoller Weise ganzzahlig und kleiner als der Range sein. (‡)

Hinweise:
- mu und sigma werden nur benötigt, falls *priortype == 2*. Sonst genügen Dummy-Werte.
- ConCoeff und CI_max müssen nur übergeben werden, falls *termtype ==* 0.
- den Anfangswert der Reizstärke erhält man, indem man die Funktion mit leeren Vektoren x_hist=[] und r_hist=[] aufruft. Dieser wird in x_next zurückgegeben.

Quellen:

Ciba, S. (2008). Erstellung einer Softwarebibliothek für Hörversuche – Programmkonzept und zu implementierende Testverfahren. Magisterarbeit, FG Audiokommunikation, Institut für Sprache und Kommunikation, Fakultät I, TU Berlin.

Cornsweet, T. N. (1962). The Staircase-Method in Psychophysics. American Journal of Psychology, 75, S. 485–491.

Findlay, J. M. (1978). Estimates on probability functions: A more virulent PEST. Perception & Psychophysics, 23, Nr. 2, S. 181–185.

King-Smith, P. E.; Grigsby, S. S.; Vingrys, A. J.; Benes, S. C.; Supowit, A. (1994). Efficient and Unbiased Modifications of the QUEST Threshold Method: Theory, Simulations, Experimental Evaluation and Practical Implementation. Vision Research, 34, Nr. 7, S. 885–912.

Levitt, H. (1971). Transformed up-down methods in psychoacoustics. Journal of the Acoustical Society of America, 49, Nr. 2, S. 467–477.

Lieberman, H.; Pentland, A. (1982). Microcomputer-based estimation of psychophysical thresholds: The Best PEST. Behavioral Research Methods & Instrumentation, 14, Nr. 1, S. 21–25.

Taylor, M. M.; Creelman, C. (1967). PEST: Efficient estimates on probability functions. Journal of the Acoustical Society of America, 41, Nr. 4, S. 782–787.

Treutwein, B. (1995). Adaptive Psychophysical Procedures. Vision Research, 35, Nr. 17, S. 2503–2522.

Treutwein, B. (1997). YAAP: yet another adaptive procedure. Spatial Vision, 11, Nr. 1, S. 129–134.

Watson, A. B.; Pelli, D. (1983). QUEST: A Bayesian adaptive psychometric method. Perception & Psychophysics, 33, Nr. 2, S. 113–120.

II. Plot-Funktionen welche in die grafische Benutzeroberfläche eingebunden wurden:

plot_psyfunc.m

Die Funktion plottet den Verlauf der psychometrischen Funktion unter Annahme einer logistischen Funktion. (Die Lage ist dabei auf die Mitte des Range fixiert.)

range	Integer	Legt den Bereich [1, *range*] möglicher Reizdarbietungen (Range) fest, welcher in Schritten von 1 aufgelöst wird.
n_or_gamma	Integer/Double	Festlegung des Antwortparadigmas (Werte größer oder gleich 2: Anzahl n der Intervalle beim nAFC-Paradigma, Werte aus [0, 1]: False-Alarm-Rate gamma für ja/nein-Paradigma).
beta	Double	Spread-Parameter.
lambda	Double	Lapsing-Rate (aus [0, 1-*gamma*]).

plot_prior.m

Die Funktion plottet die A-priori-Likelihood-Funktion für das ML/Bayes-Verfahren für drei verschiedene Varianten der Initialisierung.

range	Integer	siehe MLBayes.m
n_or_gamma	Integer/Double	
priortype	Integer	

beta	Double	
lambda	Double	

Zusätzlich für *priortype* == 2:

mu	Double	siehe MLBayes.m
sigma	Double	