Contents

1 Description

The SAVE framework is implemented in MATLAB [1] and based on the Psychtoolbox-3 [2], it is required to install both beforehand. In general, it is a framework for visual-evoked potential (VEP) and event-related potential (ERP) experiments. Among the predefined layouts and stimulation patterns, it offers an interface to easily build presentation layouts with freely placeable targets and an interface to create arbitrary stimulation patterns for each target of the layout. Those interfaces will be explained in section 5 and 6.

The stimulation length is bound to the refresh rate of the used monitor and can be defined at the beginning of the experiment, further details to the parameter will be explained in section 3.2. Since those experiments require strict stimuli timings, it also take frame drops into account and compensate them by informing the stimulation interface about the number of recent frame drops, if any occur. Furthermore, it also takes the monitor raster latency into account by offering information on where each target is vertically placed on the screen in order to easily correct it.

The application is split into two modules communicating over TCP/IP. One module responsible for the experiment itself and one to remotely operating the experiment (set parameters, choose targets, etc.). At each frame build, the operator module will get information about how each target is modulated. The synchronization with an EEG amplifier is done by the parallel port by giving information of the current state (running or not, trial, inter-trial, etc.), further details in section 4.8.

The framework was tested on Microsoft© Windows 10™, but should work on Linux and Apple© macOS™with some slight modifications (like parallel port control).

2 Initialization

The application is split into two modules communicating over TCP/IP. One module responsible for the experiment itself and one to remotely operating the experiment. For both modules an objects has to be initialized, whereby the experiment object provides the TCP/IP server and should be initialized first.

2.1 Experiment object

To initialize the experiment object, only one parameter is required, the screen number on which the window should show up. If only one monitor is connected

to the personal computer (PC), the screen-number is always 0, otherwise it is a positive number (1, 2, etc.). But, to avoid synchronization errors, it is recommended to only use one single monitor.

```
% create experiment object and show window on screenNumber
screenNumber = 0;
exp_obj = vep_experiment(screenNumber);
```

2.1.1 Optional parameters

- **tcpip** - enable (true) or disable (false) the TCP/IP Server (default: true). Disabling the TCP/IP server only makes sense for offline experiments, because you can't perform commands (like choosing a target) while experiment is running.
- **port** - the port on which the server is listening (default: 3000)
- **settings** - a struct with experimental settings (see section 3.2)
- **debug** - prints debug messages to console if true (default: false)

```
% create experiment object with optional parameters
screenNumber = 0;
settings = struct(); % empty struct will use default settings
exp_obj = vep_experiment(screenNumber,'tcpip',false,'settings',settings,'debug',true);
```

2.2 Operator object

To initialize the operator object, it is required to previously initialize the experiment object with enabled TCP/IP server. The only required parameter is HostAndPort, which is a string of the IP address and the port number, separated by ":".

```
% create operator object and connect to experiment object
HostAndPort = '127.0.0.1:3000';
op_obj = vep_operator(HostAndPort);
```

2.2.1 Optional parameters

- **settings** - a struct with experimental settings (see section 3.2)
- **debug** - prints debug messages to console if true (default: false)

```
% create operator object with optional parameters
HostAndPort = '127.0.0.1:3000';
settings = struct(); % empty struct will use default
    settings
op_obj = vep_operator(HostAndPort,'settings',settings,'
    debug',true);
```

3 Experimental Parameters

This section explains how to set experimental parameters, how to start an experiment, and how to interact (like choosing a target) with the experiment. There is also a full example in section 7

3.1 How to set parameters?

Experimental parameters can be set during initialization, as described in section 2 or by calling the "set" function of the experiment object or the operator object. The function accepts multiple key/value pairs in order to set a single or multiple parameters at once, whereas a key is always a string. Furthermore, the function accepts a struct of key/value pairs

```
% set experimental parameters by using key/value pairs
op_obj.set('parameter_name1',parameter_value1,'
    parameter_name2',parameter_value2,...);
% set experimental parameters by using a struct
settings = struct();
settings.parameter_name1 = parameter_value1;
settings.parameter_name2 = parameter_value2;
op_obj.set(settings);
```

3.2 Possible parameters

- **monitorResolution** - the monitor resolution that should be set (default: [1920,1080])

- **monitorRefreshRate** - the monitor refresh rate that should be set (default: 60)
- **windowSize** - the dimension of the window ([top,left,width,height] in pixels or 'fullscreen', default: 'fullscreen')
- **hideCursor** - hides the cursor if true (default: true)
- **layout** - name of the layout to be used for the experiment (default: 'keyboard_old' - see section 5)
- **layoutSettings** - struct containing the layout settings. For example, layoutSettings.infoColor is the RGBA font color of the info text. (see section 5)
- **stimulation** - name of the stimulation type used for the experiment (default: 'cvep' - see section 6)
- **stimSettings** - struct containing the stimulation settings (see section 6)
- **framesPerStimulus** - number of frames for a single stimulus (default: 1)
- **trials** - array of target indices used as trial order (default: 1)
- **freeMode** - endless run if true, otherwise stops after trials (default: false)
- **asynchronous** - asynchronous mode (true) or synchronous (false) (default: false)
- **startWait** - time (in seconds) to wait before a run will be started (default: 5)
- **trialTime** - time (in seconds) of each trial (synchronous mode) or minimum trial time (asynchronous mode) (default: 1.05)
- **interTrialTime** - time (in seconds) between 2 successive trials (default: 1.05)
- **playbackMode** - enables the playback mode (if true, don't set parallel port and don't send current target modulation through TCP/IP)

4 Experimental functions

This section will explain how to start and how to interact with the experiment. For an offline experiment it is only required to set experimental settings, start the experiment and wait until it is finished. This can be done without using the operator object. For an online experiment, the operator object is mandatory. For simplicity both the experiment object and the operator object have the same function names for the most important functions.

4.1 How to start the experiment?

After all settings are set, the experiment can be started by calling the "start" function.

```
% start experiment using the experiment object
exp_obj.start();
% OR start experiment using the operator object
op_obj.start();
```

4.2 How to interrupt the experiment?

You are using the free mode or something went wrong and you want to stop the experiment? This can be done by the operator object, calling the "stop" function.

```
% interrupt the experiment
op_obj.stop();
```

4.3 How to show info text to the participant?

You want to introduce the participant or want to show information after the end of a trial/run? This can be done by calling the "info" function of the operator object.

```
% show info text
text = 'Accuracy: ...';
duration = 10; % show text for duration seconds,
    optional
op_obj.info(text,duration);
```

4.4 How to highlight a target?

You want to introduce the participant at which target he has to look at? This can be done by calling the "highlightTarget" function of the operator object.

```
% highlight target 1 for 10 seconds
targetIdx = 1; % the index of the target, see layout
    section
duration = 10; % highlight the target for duration
    seconds
op_obj.highlightTarget(text,duration);
```

4.5 How to choose a target?

Your classifier predicted a target and you want to choose it? This can be done by calling the "chooseTarget" function of the operator object.

```
% choose target 1 and highlight it for 10 seconds
targetIdx = 1; % the index of the target, see layout
    section
duration = 10; % highlight the target for duration
    seconds, optional
op_obj.chooseTarget(targetIdx,duration);
```

4.6 How to disconnect TCP/IP connection?

You want do disconnect the TCP/IP connection? This can be done by calling the "disconnect" function of the operator object, after that the server will be reinitialized.

```
% disconnect TCP/IP connection and reinitialize the
    server
op_obj.disconnect();
```

4.7 Other functions

There are some more functions, which are not explained here. Have a look at the code itself, everything is commented. In general, only functions declared as "public" are relevant to perform an experiment.

4.8 Synchronization

The synchronization is done using the parallel port.

- 1. bit - is 1 if a run has started, 0 otherwise
- 2. bit - is the stimulation pattern of the first target, this can be used to check if the stimulation pattern is as expected.
- 3. bit - toggles between 1 and 0 for each frame
- 4. bit - is 1 during a trial and 0 otherwise
- 5. bit - can be set dynamically for each stimulation pattern, for example you can mark the start of sequence if you use a cyclic stimulation pattern (see section 6.3.4)

In order to write to the parallel port, you need the correct libraries for your OS and MATLAB version. To keep things easier we wrote a simple wrapper class `PPort` which accepts the address of the parallel port as decimal value or as a binary string. The class is currently only working for windows but independent of the architecture and the MATLAB version (x86 or x64). At the first run it is required to run MATLAB as administrator in order to install the libraries.
The `PPort` class makes use of the Open Source libraries *InpOut32* and *InpOutx64* (© Logix4U & Phillip Gibbons). Further informations can be found here:
`http://www.highrez.co.uk/Downloads/InpOut32/default.htm`
and here:
`http://apps.usd.edu/coglab/psyc770/IO64.html`

5 Layouts

There are some predefined layouts which will be explained in the following subsection. If you want to create your own layout, see subsection 5.2. Parameters for a layout are defined using a struct which in turn must be set as a the value for the global parameter *layoutSettings* (see 3.2).

5.1 Predefined Layouts

5.1.1 Keyboard matrix

The keyboard matrix is named **keyboard** and has a definable number of rows and columns. The labels of each target can also be defined.

Parameters

- **targetColor** - RGBA color of the target label font (default: gray)
- **highlightColor** - RGBA color used for highlighting a target (default: yellow)
- **stimulusColor** - RGBA color used for stimulation (default: white)
- **boxes_x** - number of columns (default: 8)
- **boxes_y** - number of rows (default: 4)
- **target_names** - cell containing one entry for each target (length = boxes_x×boxes_y). (default: cellstr(['A':'Z','_','1':'5']))

5.1.2 Single Target

There is also very simple layout named **single**, containing only one centered square target without any label. This layout is suitable for testing purposes and to have a minimal example on how to create new layouts.

Parameters

- **highlightColor** - RGBA color used for highlighting a target (default: yellow)
- **stimulusColor** - RGBA color used for stimulation (default: white)
- **width** - width of the target in pixels (default: 150)
- **height** - height of the target in pixels (default: 150)

5.2 How to create a new layout?

It is really easy to create new layouts. For this we implemented an abstract class 'screenlayout.m' from which each layout has to inherit. Furthermore, the layout must be named 'screenlayout_LAYOUTNAME.m' and placed in the folder 'layouts'.

5.2.1 Constructor

The constructor of your layout class must call the constructor of the superclass, this is mandatory! Minimal example:

```
function this = screenlayout_LAYOUTNAME(windowPtr,
    varargin)
  this@screenlayout(windowPtr, varargin{:});
end
```

5.2.2 Define parameters

If you want to use definable parameters, use the *inputParser* of MATLAB:

```
function this = screenlayout_LAYOUTNAME(windowPtr,
    varargin)
  this@screenlayout(windowPtr, varargin{:});
  p = inputParser;
  p.StructExpand = true;
  p.KeepUnmatched = true;
  addParameter(p,...);
  ...
  parse(p, varargin{:});
end
```

See MATLAB help of *addParameter* or look at the code of one of the predefined layouts. Your parsed parameters are in 'p.Results'. The defined parameters can be set as described earlier.

5.2.3 Add targets

Targets can be added using the superclass function `addTarget(type,position,colorOrTexture,toggleColorOrTexture)`.
Where *type* can be one of 'FillRect' (rectangular target), 'FillOval' (oval target), or 'DrawTexture' (textured target, for example a picture). The

parameter *position* is [x;y;width;height] (or multiple columns for multiple targets). In case of 'FillRect' or 'FillOval' the parameters *colorOrTexture/toggleColorOrTexture* are [r;g;b;a] (or multiple columns for multiple targets) or in case of 'DrawTexture' they must be the index of a texture (see `http://docs.psychtoolbox.org/MakeTexture`). The difference between *colorOrTexture* and *toggleColorOrTexture* is that the first is for the on-stimulus and the latter for the off-stimulus.

If you want to add labels to the target use the function `setTargetNames(names,color,fontsizes)`, where *names* has to be a cell containing the labels for each target, *color* is [r;g;b;a] (one column for each target, or one column for all targets) and *fontsizes* is a list of fontsizes (one for each target).

5.2.4 Add textfields

The superclass also offers function for textfields. You can add multiple textfields and set/get the text of them:

- `addTextField(position,textcolor,backgroundcolor)` - adds a text field to the layout, where position is [x;y;width;height], textcolor is [r;g;b;a] and backgroundcolor is [r;g;b;a]. The function returns the index of the text field. You can add multiple text fields.
- `setText(idx,text)` - sets the text of a text field with index idx.
- `getText(idx)` - returns the current text of a text field with index idx.

5.2.5 Default attributes

Furthermore it offers a few attributes:

- **windowSize** - struct with fields 'width' and 'height' that store the size of window in pixels and should be used to create scalable layouts.
- **numTargets** - the number of targets (at runtime).
- **windowPtr** - the window pointer of the PSYCHTOOLBOX. This can be used to draw additional non-target things.

5.2.6 Advanced Functions

The superclass also offers some advanced function to create a layout, you must override those methods in your layout class.

- `chooseTarget(index)` - defines what to do if a target with index has been chosen (used for online experiments)
- `startRun()` - this function is called once at the beginning of a run
- `startTrial()` - this function is called once at the beginning of each trial
- `intraTrial()` - this function is called before each frame draw during a trial
- `interTrial()` - this function is called before each frame draw during the inter trial time
- `endTrial()` - this function is called once at the end of each trial

5.2.7 Minimal example

```
classdef screenlayout_LAYOUTNAME < screenlayout
  methods
    function this = screenlayout_LAYOUTNAME(windowPtr,
       varargin)
      % call the superclass constructor (this is
         mandatory)
      this@screenlayout(windowPtr,varargin{:});
      % place a single target in top left corner of size
          20 x 20 pixels
      this.addTarget([0;0;20;20]);
    end
  end
end
```

6 Stimulation

There are some predefined stimulation patterns which will be explained in the following subsection. If you want to create your own stimulation pattern, see subsection 6.3.

6.1 Parameters

Parameters for a stimulation pattern are defined using a struct which in turn must be set as a the value for the global parameter *stimSettings* (see 3.2). There are some parameters used for all stimulation pattern:

- **interTrialStimuli** - defines if the target should flicker during the inter trial time (true), or not (false). (default: false)

- **fixFrameDrops** - defines if the 'stimPos' attribute should be increased by the number of occurred frame drops, this can be used to reduce the effects of frame drops by skipping frames which should theoretically be presented. (default: true)

- **syncAtTrialStart** - defines if the 'stimPos' should be reset at trial start (true) or not (false). (default: true)

6.2 Predefined stimulation patterns

6.2.1 cVEP pattern

The stimulation pattern 'cvep' makes use of an m-sequence with low auto-correlation. The sequence will be shifted by a definable amount of bits for each successive target. Furthermore, you can vary the m-sequence. The code 'mseq.m' for m-sequence generation is from Buračas and Boynton [3].

Parameters

- **mseqParams** - parameters used for m-sequence generation (default: [2,6,1,1] - see mseq.m-File)

- **mseqShift** - number of bits the m-sequence should be shifted for successive targets (default: 2)

6.2.2 Fully random stimulation pattern

Using the stimulation pattern **random**, the target stimulation will be modulated with a fully random sequence. The seed of the random generator is definable using the parameter **randomseed**.

6.2.3 SSVEP stimulation pattern

The stimulation pattern 'ssvep' allows to modulated targets with a binary or sinusoidal stimulation pattern with a definable frequencies and phase-shifts. It automatically takes the monitor refresh rate into account, in order that the defined frequency will be real stimulation frequency.

Parameters

- **frequency** - stimulation frequency of each single target or one for all targets. (default: 1 Hz for each target)
- **phaseshift** - phase-shift of each target, 0 means stimulation starts with full illuminated on-stimulus. Rembember: a full phase is 2π. (default: 0 for each target)
- **binary** - binary stimulation (true) meaning only full illuminated on- or off-stimului, or sinusoidal stimulation (false) meaning the on- and off-stimuli will blend into each other. (default: false)

6.2.4 Predefined stimulation for each target

This class allows to define the stimulation sequence using a CSV-file. If the number of targets equals the number of rows in the CSV-file, then the sequences will be assigned to the targets in ascending order and will be processed circularly. If there are more rows than the targets, the sequences will be assigned randomly to the targets, ensuring that each two targets will not have the same sequence assigned. Once the sequences are processed, new sequences will be assigned.

Parameters

- **sequencePool** - path to the CSV-file
- **randomseed** - the seed for the random generator used for random assignment

6.3 How to create a new stimulation pattern?

Like for the layouts, it is really easy to create new stimulation patterns. For this we implemented an abstract class 'stimulation.m' from which each stimulation pattern has to inherit. Furthermore, the stimulation pattern must be named 'stimulation_STIMULATIONNAME.m' and placed in the folder 'stimulations'.

6.3.1 Constructor

The constructor of your stimulation class must call the constructor of the superclass, this is mandatory! Minimal example:

```
function this = stimulation_STIMULATIONNAME(numTargets,
    varargin)
  this@stimulation(numTargets,varargin{:});
end
```

6.3.2 Define parameters

If you want to use definable parameters, use the *inputParser* of MATLAB:

```
function this = stimulation_STIMULATIONNAME(numTargets,
    varargin)
  this@stimulation(numTargets,varargin{:});
  p = inputParser;
  p.StructExpand = true;
  p.KeepUnmatched = true;
  addParameter(p,...);
  ...
  parse(p,varargin{:});
end
```

See MATLAB help of *addParameter* or look at the code of one of the predefined stimulation patterns. Your parsed parameters are in 'p.Results'. The defined parameters can be set as described earlier.

6.3.3 Default attributes

There are a few attributes that could be required to create your stimulation pattern:

- **numTargets** - this is the number of targets of the current layout
- **numBits** - this is the total number of processed frames/bits DURING the trials of the current run
- **trialPos** - this is the number of processed frames/bits of the current trial
- **stimPos** - this SHOULD be used for your stimulation pattern, it is either **trialPos** or **numBits**, depending on the parameter **syncAtTrialStart** (see section 6.1)
- **isTrial** - this is boolean value indication if `next` is called during a trial or an inter-trial
- **numTrials** - this is the number of completed trials of the current run.

6.3.4 Functions

- `next(lostFrames)` - this function is called at each frame refresh and is required to return the opacity (alpha value) for each single target for the next frame. In order to know the number of targets of the current layout, there is the attribute field 'numTargets'. The 'lostFrames' parameter gives information of how many frames were lost (frame drops) during the last call, it will automatically increase the **stimPos** attribute (see section 6.1) in order to ensure synchronization. It is mandatory to call the superclass method: `next@stimulation(this,lostFrames)`.

- `isSequenceStart()` - this function is optional. If used it has to return a single boolean value. For example, in case of 'cvep' stimulation it is set to true at each start of the m-sequence sequence. The value will be set as the fifth bit parallel port

6.3.5 Minimal example

In the following is minimal example on how to create a stimulation pattern:

```
classdef stimulation_STIMULATIONNAME < stimulation
  methods
    function this = stimulation_STIMULATIONNAME(
        numTargets, varargin)
      % call the superclass constructor (this is
          mandatory)
      this@stimulation(numTargets, varargin{:});
    end

    function alpha = next(this, lostFrames)
      % call the superclass function (this is mandatory)
      next@stimulation(this, lostFrames);
      % return a random integer (0 or 1) for each single
          target -> random stimulation
      alpha = this.randstream.randi(2,1,this.numTargets)
          -1;
    end
  end
end
```

7 Example

7.1 Simple example without operator

The layout will be the 8×4 keyboard matrix with cVEP stimulation. The subject has to perform 32 trials starting with target 1 and ending with target 32. The monitor resolution and refresh rate will be set to 1920×1080 and 60 Hz, respectively. The window will be shown in full-screen mode and the mouse cursor will be hidden. Both the trial length and the inter-trial time lasts for 1.05 seconds, which is exactly the required time to show a full cycle of the m-sequence at 60 Hz. After calling the "start" function, the participant will see a counter for 10 seconds until the experiment will be started. Once the experiment is finished, it will automatically close the window.

```
% Ensure the workspace is clear and open screens are
    closed
sca;
close all;
clearvars;
% add required paths
addpath('vepstim','inpout');

% Returns all available screen indizes
screens = Screen('Screens');
% Choose screen with max index
screenNumber = max(screens);

% define some experimental settings (see section 3.2)
settings = struct();
settings.monitorResolution = [1920,1080];
settings.monitorRefreshRate = 60;
settings.stimulation = 'cvep';
settings.layout = 'keyboard';
settings.windowSize = 'fullscreen';
settings.startWait = 10;
settings.trialTime = 1.05;
settings.interTrialTime = 1.05;
settings.trials = 1:32;
settings.hideCursor = true;

% initialize experiment object without TCP/IP server and
    debug disabled
exp_obj = vep_experiment(screenNumber,'tcpip',false,'
```

```
    settings',settings,'debug',false);
28
29 % start the experiment, blocks until finished
30 exp_obj.start();
31
32 % Close the window after experiment.
33 exp_obj.exit();
```

7.2 Simple example with operator

This is a similar example, but using the operator. With the operator it is possible to interact with experiment while running, like choosing some targets (writing some letters in this case).
Running on PC1 (IP: 192.168.1.1):

```
1  % Ensure the workspace is clear and open screens are
       closed
2  sca;
3  close all;
4  clearvars;
5  % add required paths
6  addpath('vepstim','inpout');
7
8  % Returns all available screen indizes
9  screens = Screen('Screens');
10 % Choose screen with max index
11 screenNumber = max(screens);
12
13 % initialize experiment object with TCP/IP server
       listening at port 3000
14 exp_obj = vep_experiment(screenNumber);
```

Running on PC2 (IP: 192.168.1.2):

```
1  % Ensure the workspace is clear
2  clearvars;
3  % add required paths
4  addpath('vepstim');
5
6  % define some experimental settings (see section 3.2)
7  settings = struct();
8  settings.monitorResolution = [1920,1080];
9  settings.monitorRefreshRate = 60;
```

```
10 settings.stimulation = 'cvep';
11 settings.layout = 'keyboard';
12 settings.windowSize = 'fullscreen';
13 settings.startWait = 10;
14 settings.trialTime = 1.05;
15 settings.interTrialTime = 1.05;
16 settings.trials = 1:32;
17 settings.hideCursor = true;
18
19 % initialize operator object with debug disabled
20 op_obj = vep_operator('192.168.1.1:3000','settings',
       settings,'debug',false);
21
22 % start the experiment, doesn't block!
23 op_obj.start();
24
25 % now you can send some commands, like choosing a target
       (of course, this should be done by your classifier)
26 % first wait 15 seconds
27 pause(15);
28
29 % Now write some letters and highlight target for 100 ms
       . Wait 2 seconds after each.
30 op_obj.chooseTarget(8,0.1); pause(2);
31 op_obj.chooseTarget(1,0.1); pause(2);
32 op_obj.chooseTarget(12,0.1); pause(2);
33 op_obj.chooseTarget(12,0.1); pause(2);
34 op_obj.chooseTarget(15,0.1); pause(2);
35
36 % terminate experiment and close window
37 op_obj.exit()
```

References

[1] MATLAB, version 9.1 (R2016b), The MathWorks Inc., Natick, Massachusetts, 2016.

[2] D. H. Brainard, S. Vision, The psychophysics toolbox, Spatial vision 10 (1997) 433–436.

[3] G. T. Buračas, G. M. Boynton, Efficient design of event-related fmri experiments using m-sequences, Neuroimage 16 (3) (2002) 801–813.