OPI – Open Perimeter Interface
Version 1.1

Contents

1 Preamble

This document describes a standard set of R functions and classes for interfacing with a perimeter (an instrument for examining visual fields). It began existence at the First Octopus Research Meeting held in Tübingen in July 2010, which was hosted by Prof. Ulrich Schiefer (University of Tübingen), and Matthias Monhart (Haag-Streit). R code that implements this interface should provide the set of functions described.

1.1 Document History

0.0	3 Jul 2010	Began in Hotel Hospiz, Tübingen by Andrew Turpin.
0.1	7 Jul 2010	Complete rewrite with feedback from Janko Dietzsch and Shaban Demirel.
0.2	17 May 2011	Redraft by Andrew Turpin based on extensive feedback over a few rounds from Paul Artes and Shaban Demirel.
0.3	9 July 2011	`image` added to stimuli and a few errors in examples fixed by Turpin.
0.4	19 September 2011	Turpin added "next" to opiPresent (to allow efficiencies on projection systems) and changed arguments to `opiPresent` to lists.
0.5	20 September 2011	Turpin on advice from Jim Cassidy added `opiQueryDevice` function.
1.0	16 September 2014	After several companies expressed interest in supporting the OPI on their perimeter during IPS 2014 (New York), and versions for the HEP and Octopus 900 need to be revised, Turpin added the first Appendix describing a server side version to support the OPI.
1.1	1 July 2017	The advent of real-time eye tracking in perimeters means that `opiInitialize` and `opiClose` can now return meaningful data. After discussions with Alberto Rosso (CenterVue) at WGC 2017 in Helsinki, Turpin has altered the return types for these commands to a list, which is backwards compatible.

1.2 Document Future

It is expected that this document will be revised at meetings of the Imaging and Perimetry Society.

1.3 Conventions

All (x, y) coordinates are Cartesian relative to the fixation point $(0, 0)$ in degrees of visual angle (not radians). Positive y-coordinates refer to stimuli in the superior field. Positive x-coordinates refer to locations "east" of fixation (temporal for the Right eye, nasal for the Left).

For "traditional" perimetry, where stimuli are projected spots, the diameter of the spot is in degrees of visual angle, hence a Goldmann Size III would have $size = 0.43$. The luminance of the spot is in cd/m^2, thus the total luminance on a projection system is this plus the background luminance, while on a CRT system, the spot would "override" the background luminance. Spot luminance is not specified in dB or apostilbs, etc.

When the stimuli is an image, *size* becomes a scaling parameter and luminance is specified in the image itself.

All times are in milliseconds.

1.4 Usage

An implementation of this interface as a minimum should support the basic types and functions below. It may be the case that specific implementations provide more functionality than that described here, and that is great! For more sophisticated stimuli (eg ramping of a static stimulus), subclass the existing types and write your own presentation methods.

2 Data Types

opiStaticStimulus

```
stim <- list(x, y, image=NA, level, size=0.43, color="white",
             duration=200, responseWindow=1500, ...)
class(stim) <- "opiStaticStimulus"
```

x	x coordinate of the center of stimulus in degrees relative to fixation
y	y coordinate of the center of stimulus in degrees relative to fixation
image	an image to display in a machine specific format
level	stimulus level in cd/m^2 (ignored if `!is.na(image)`)
size	diameter of target in degrees, or scaling factor for `image` if specified
color	machine specific stimulus color settings (ignored if `!is.na(image)`)
duration	total stimulus duration in milliseconds
responseWindow	maximum time (>= 0) in milliseconds to wait for a response from the onset of the stimulus presentation
...	machine specific parameters

opiTemporalStimulus

```
stim <- list(x, y, image=NA, lut, size=0.43, color="white", rate, duration,
             responseWindow=1500, ...)
class(stim) <- "opiTemporalStimulus"
```

x	x coordinate of the center of stimulus in degrees
y	y coordinate of the center of stimulus in degrees
image	TRUE for `lut` containing images, `FALSE` for luminances
lut	if `image` is `FALSE` then this is a lookup table (vector) for stimulus level at each step of `rate` Hz in cd/m^2. If `image` is `TRUE`, then this is a list of images, in the same format as `image`, that is stepped through at `rate` Hz.
size	diameter of target in degrees, or scaling factor for images
color	machine specific stimulus color settings (ignored if `image`)
rate	frequency with which `lut` is processed in Hz
duration	total length of stimulus flash in milliseconds. There is no guarantee that `duration` mod $\|$`lut`$\|$/`rate` == 0. That is, the onus is on the user to ensure the duration is a multiple of the period of the stimuli.
responseWindow	maximum time (>= 0) in milliseconds to wait for a response from the onset of the stimulus presentation
...	machine specific parameters

opiKineticStimulus

```
stim <- list(path, images=NA, levels, sizes, colors, speeds, ...)
class(stim) <- "opiKineticStimulus"
```

path	list of (x,y) coordinates in degrees that is usable by `xy.coords()`
image	`image[i]` is the image to display (in a machine specific format) in the section of the path specified by `path[i]..path[i+1]`.
levels	if `is.na(image)` then `levels[i]` is the stimulus level in cd/m^2 in the section of the path specified by `path[i]..path[i+1]`.
sizes	`sizes[i]` is the size of stimulus (diameter in degrees) to use for the section of path specified by `path[i]..path[i+1]`, or a scaling factor for `images[i]`.
colors	`colors[i]` is the color to use for the stimulus in the section of path specified by `path[i]..path[i+1]`. Machine specific values. Ignored if `!is.na(image)`.
speeds	`speeds[i]` is the speed (degrees per second) for the stimulus to traverse the path specified by `path[i]..path[i+1]`.
...	machine specific parameters

3 Functions

For functions that use machine specific parameters, it is recommended that you prefix your parameter names with a unique, machine specific code so that the function can be called without alteration on several implementing machines.

opiInitialize(...)

Arguments:

... machine specific parameters.

Description:

This function specifies any machine specific parameters that are necessary to get the machine into a state for accepting further commands. It must be called before any other OPI functions. If it is not called, the behaviour of all other OPI functions are not defined.

Value:

`opiInitialize` returns a list containing at least the following component. Machine specific implementations may have more components in the list.

err NULL if succeeded without error, machine specific error code otherwise.

Note that for backwards compatibility, it is also acceptable for this function to return NULL when there is no error.

Examples:

```
ret <- opiInitialize("SL")  # Put the HFA I into Slave mode
if (!is.null(ret) && !is.null(ret$err))
    stop(paste("OPI Error initializing machine:",ret$err))
```

opiSetBackground(lum, color, ...)

Arguments:

lum	Set background illumination to lum cd/m^2. lum= 0 is no background illumination.
color	machine specific background color
...	machine specific parameters.

Description:

This function sets the background of the perimeter.

Value:

error	NULL if succeeded without error, machine specific error code otherwise.

Examples:

```
ret <- opiInitialize()
if (!is.null(ret) && !is.null(ret$err))
    stop(paste("OPI Error initializing machine:",ret$err))

opiSetBackground(100, "yellow") # SWAP (blue-on-yellow) background

opiSetBackground(10,  "white")  # HFA white-on-white background

err <- opiSetBackground(10,  "white")
if (!is.null(err))
    stop(paste("OPI Error setting background:",err))
```

opiPresent(stim, ...)

Arguments:

stim	a list of instances of `opiStaticStimulus`, `opiTemporalStimulus`, or `opiKineticStimulus`. If `stim` is NULL then the machine simply returns its status (as defined by the machine) in the `err` field and does not present any stimuli.
next=NULL	a list of instances of `opiStaticStimulus`, `opiTemporalStimulus`, or `opiKineticStimulus` that are the stimuli to present after `stim`.
...	machine specific parameters.

Description:

Generic function for presentation of stimulus `stim`. Should contain implementations for the three possible classes of stim:

- `opiPresent.opiStaticStimulus(stim, ...)`,
- `opiPresent.opiTemporalStimulus(stim, ...)`, and
- `opiPresent.opiKineticStimulus(stim, ...)`.

`opiPresent` is "blocking" in that it will not return until either a response is obtained, or at least the `responseWindow` milliseconds has expired. (Note that more time might have expired.) Specifying next allows the implementing machine to use the time waiting for a response to `stim` to make preparations for the `next` stimuli. (For example retargeting the projector or moving aperture and/or filter wheels.) There is no guarantee that the next call to `opiPresent` will have `next` as the first argument; this should be checked by the machine specific implementations.

Value:

`opiPresent` returns a list containing at least the following components. Machine specific implementations may add components to this list.

err	NULL if no error occurred, otherwise a machine specific error message. This should include errors when the specified size cannot be achieved by the device (for example, in a projection system with an aperture wheel of predefined sizes.) If `stim` is NULL, then `err` contains the status of the machine.
seen	TRUE if a response was detected in the allowed `responseWindow`, FALSE otherwise.
time	The time in milliseconds from the onset of the presentation until response from the subject if `seen` is TRUE. If `seen` is FALSE, this value is undefined.

Examples:

```
ret <- opiInitialize()
if (!is.null(ret) && !is.null(ret$err))
    stop(paste("OPI Error initializing machine:",ret$err))

    # HFA white-on-white background and Goldmann Size III 10dB stimulus
    # 10dB == 1000 aps == 318.3 cd/m^2
    # BUG? check is it 318 - 10 for the background?
opiSetBackground(10,  "white")
stim <- list(x=-3, y=-3, level=318, size=0.43, color="white",
             duration=500, responseWindow=1500)
class(stim) <- "opiStaticStimulus"
result <- opiPresent(stim)
if (!is.null(result$err))
    stop(paste("OPI Error:", result$err, "presenting", stim))
if (result$seen)
```

```
    print(paste("Saw stimulus in",result$time,"milliseconds.")
else
    print("Did not see stimulus.")

    # HFA white-on-white background and Goldmann Size III 10dB stimulus
    # at (-3,3) followed by a 10dB stimulus at (9,9)
opiSetBackground(10,  "white")
stim1 <- list(x=-3, y=-3, level=318, size=0.43, color="white",
             duration=500, responseWindow=1500)
stim2 <- list(x=9, y=0, level=318, size=0.43, color="white",
             duration=500, responseWindow=1500)
class(stim1) <- "opiStaticStimulus"
class(stim2) <- "opiStaticStimulus"
result <- opiPresent(stim1, stim2)
if (!is.null(result$err))
    stop(paste("OPI Error:", result$err, "presenting", stim1))
if (result$seen)
    print(paste("Saw stimulus in",result$time,"milliseconds.")
else
    print("Did not see stimulus.")
result <- opiPresent(stim2)
if (!is.null(result$err))
    stop(paste("OPI Error:", result$err, "presenting", stim2))
if (result$seen)
    print(paste("Saw stimulus in",result$time,"milliseconds.")
else
    print("Did not see stimulus.")


    # A Size III white kinetic stimuli on
    # a bilinear path {(27,27), (15,20), (0,0)}
stim <- list(path=list(x=c(27,15,0), y=c(27,20,0)),
             sizes=rep(0.43,2),
             colors=rep("white",2),
             levels=rep(318,2),
             speeds=c(4,3))
class(stim) <- "opiKineticStimulus"
result <- opiPresent(stim)
if (!is.null(result$err))
    stop(paste("OPI Error:", result$err, "presenting", stim))
if (result$seen)
    print(paste("Saw stimulus in",result$time,"milliseconds.")
else
    print("Did not see stimulus.")

    # A Size III flickering with a 10Hz square wave at
    # location (7,7) with luminance 10 dB (HFA)
stim <- list(x=7, y=7, size=0.43, color="white",
             rate=20,        # one lut step per 50 ms
             lut=c(0,318),   # so one full lut per 100 ms == 10Hz
             duration=400,   # and 4 cycles per stimulus
             responseWindow=1500)
class(stim) <- "opiTemporalStimulus"
result <- opiPresent(stim)
if (!is.null(result$err))
    stop(paste("OPI Error:", result$err, "presenting", stim))
```

```
if (result$seen)
    print(paste("Saw stimulus in",result$time,"milliseconds.")
else
    print("Did not see stimulus.")

    # An FDT patch at location (7,7)
    # with luminance 10 dB (HFA units)
## TODO

opiClose()
```

opiClose(...)

Arguments:

... machine specific parameters for ending a session.

Description:

Close the session, perhaps returning the machine to its normal state.

Value:

`opiClose` returns a list containing at least the following component. Machine specific implementations may have more components in the list.

err NULL if succeeded without error, machine specific error code otherwise.

Note that for backwards compatibility, it is also acceptable for this function to return NULL when there is no error.

Examples:

```
ret <- opiInitialize()
if (!is.null(ret) && !is.null(ret$err))
    stop(paste("OPI Error initializing machine:",ret$err))

...  # some other things in here

ret <- opiClose()
if (!is.null(ret) && !is.null(ret$err))
    stop(paste("OPI Error closing machine:",ret$err))
```

opiQueryDevice(...)

Arguments:

... machine specific parameters.

Description:

This function returns information about the perimeter that might be required for determining stimuli, etc. For example, the maximum and minimum x and y coordinates, the maximum brightness, the version number, etc.

Value:

... a list of machine specific values.

Examples:

```
ret <- opiInitialize()
if (!is.null(ret) && !is.null(ret$err))
    stop(paste("OPI Error initializing machine:",ret$err))
info <- opiQueryDevice()
...
```

4 Other functionality

It is expected that as perimeters evolve, new aspects of machines will be considered "standard" and will make their way into this document.

One particular innovation that is not covered in this API is that of gaze monitoring via video camera. Currently it is expected that information about gaze during a presentation will be available as part of the list of values returned by `opiPresent`.

A Server side

For new devices wishing to support the OPI, we would recommend that the device implement a simple TCP/IP socket protocol that can then be supported by the OPI R package that implements the OPI standard. This appendix outlines a simple protocol that would be required. The protocol can be enhanced to support machine specific features as required, and the details of the message data would obviously need to be specified per machine.

The messages are passed to the server as space delimited text strings of the format `command parameters`, and get a return message of the form {`OK | ERR`} `data`. The next two tables specify these two messages.

A.1 OPI Communications Protocol

Command	Param's	Description	Return data
OPI-SET-MODE	x	$x \geq 0$ indexes the stimuli types available on the machine, for example SAP, FDF, FDT, and so on. As many values as necessary can be used. The modes can also include machine dependent features such as turning eye tracking on/off, toggling automatic chin-rest support, and so on.	None
OPI-SET-FIXATION	x	x coordinate of fixation in degrees	None
	y	y coordinate of fixation in degrees	
	t	$t \geq 0$ is and index into fixation marker types supported by the machine (including "off")	None
OPI-SET-BACKGROUND	c	Machine specific color codes	None
OPI-PRESENT-STATIC	x	x coordinate of centre of stimuli in degrees	Response
	y	y coordinate of centre of stimuli in degrees	
	...	other stimulus specific parameters such as size, level, width, height, pixels (for a general image), duration, and so on	
OPI-PRESENT-KINETIC	n	number of coordinate pairs in path	Response
	x_1	first x coordinate in degrees	
	x_2	second x coordinate in degrees	
	...		
	x_n	nth x coordinate in degrees	
	y_1	first y coordinate in degrees	
	y_2	second y coordinate in degrees	
	...		
	y_n	nth y coordinate in degrees	
	...	other stimulus specific parameters such as size, level, etc.	
OPI-PRESENT-TEMPORAL	x	x coordinate of centre of stimuli in degrees	Response
	y	y coordinate of centre of stimuli in degrees	
	n	number of cycles in stimuli	
	lut1	some serialised representation of the first LUT	
	lut2	some serialised representation of the second LUT	
	...		
	lut$n$	some serialised representation of the final LUT	
	...	other stimulus specific parameters such as size, level, width, height, pixels (for a general image), duration, speed, loop-forever, and so on	
OPI-GET-DATA	d	d indicates type of data required	Data
OPI-CLOSE	None	Close the socket connection	None

Return Code	Data	Description
None	–	No data, just OK or ERR
Response	s	Seen/not-seen as a true or false
	t	If s is true, the response time in ms as determined by the machine. If s is false, this value can be anything.
	...	Perhaps gaze information or error codes if ERR, etc.
Data	x	Data specific to the call

A.2 An example session

Here is an example session on a hypothetical perimeter.

To server	From Server	Comment
`OPI-SET-MODE 0`	`OK`	Set perimeter into default SAP mode
`OPI-SET-FIXATION 0 0 2`	`ERR`	Fixation marker not allowed
`OPI-SET-FIXATION 0 0 0`	`OK`	Set cross in centre of screen
`OPI-PRESENT-STATIC -3 9 20`	`OK 1 439`	Show 20dB at position $(-3, 9)$ which is seen in 439 ms.
`OPI-PRESENT-STATIC -3 9 25`	`OK 0 -1`	Show 25dB at position $(-3, 9)$ which is not seen.
`OPI-CLOSE`	`OK`	Close socket