28 June
2019

PresLink Presentation library

MAARTEN VAN DEN HEUVEL

Table of Contents

About this library

This library contains some subroutines to use Eyelink eye trackers in Presentation, via the PresLink extension included in the SR Research developer's toolkit. The toolkit itself includes some example scripts, but these can be somewhat hard to grasp as they are not so clearly organized and/or commented. This library tries to present the tutorial functionality in a more organized and clear manner, and adds extra functionality. All the subroutines described in this document (below) are included in the .pcl-file EyelinkSUBS. Additional documentation and comments on individual lines of code are included in the pcl file. Three example scripts, based on the (Presentation course template) are provided to show different ways in which the library can be implemented. Under 'How to use this library', you can find out which of the example scripts comes closest to your own needs.

The subroutines are deliberately not presented as a template (.tem), but as a normal .pcl-file, indicating that you should feel free to change the subroutines to suit your needs (and are responsible for the proper functioning of the subroutines). Note that the defaults used in the subroutines are mostly as set in the examples provided by SR Research, but this does not mean they are necessarily always the best for your experiment.

Before you begin

Before having a look at this library, make sure The Eyelink Developers' Toolkit is installed, and PresLink is registered in Presentation. To do so, go to SR Research > Manuals > PresLink User Guide. If this entry does not exist, you probably do not have the toolkit installed and should do so by registering on the SR Research forums and downloading the 'Eyelink Developer's Kit' and (and Data Viewer if you will be using Data Viewer). Downloads are available at: https://www.sr-support.com/forum/downloads (after registering).
Even if you do have the Toolkit/Data Viewer installed, it is best to check if your version is the most recent one. Some of the subroutines listed here may not work properly with old versions. N.B.: you may need to contact someone with administrator rights to install the software and/or register the extension.

Once you have installed/found the User Guide, follow the instructions in the 'PresLinkTutorial', especially the heading 'Register PresLink with Presentation'. You can generally skip the instructions under ´Configure Your PC as a Display PC, as these settings should already be set properly. Make sure to register the extension under the exact name 'PresLink' (including correct capitalization), as some of the subroutines are programmed to use this name.

N.B.: to see a (somewhat hidden) list of all available PresLink functions, after opening the user guide, click the plus sign next to 'PresLink User Guide' on the left side of the help window, then click the subheading ´PresLink'. The 'Manuals' entry in the Start Menu should also contain a Programmers' Guide with a more extensive documentation of all the Eyelink API functions, though not all of these functions are available in PresLink.

How to use this library

Two of the most important design considerations when using PresLink are:

- Whether you want to analyze individual trials (with matching images) in Data Viewer at a later stage.

- Whether you trial screens consist of a single, pre-existing image or are generated out of several elements 'on the spot' by Presentation.

Based on these considerations, most designs can be categorized within one of the four following descriptions:

No need for Data Viewer and no need to view individual trials at a later stage:

This applies to experiment where you are not interested in the gaze paths for individual trials, but mainly need the eyetracker as an on-line check to see if users are focusing on a specific part of the screen or are looking at the screen at all. This kind of experiment is probably most similar to the 'No Trials' example script.

In this case you probably do not need any of the PresentTrial functions, as these serve mainly to get everything organized for later analysis in Data Viewer. Just start recording right after initialization (use InitializePresLink with bStartRecording set to **true**). Stop recording once your experiment ends or the eye tracking part is finished. Data will still be saved to an edf-file, but you will not be able to analyze individual trials in Data Viewer.

Need to be able to view trials in Data Viewer; trial screens are stored as single image files:

In this case the choice is also easy. Having pre-generated images saves us the trouble of having to take screenshots and deal with the resulting implications for timing . You will be presenting trials using the 'PresentWholeScreenTrialPresLink' subroutine. Having a look at the 'PreGenerated' example script should get you what you want.

Need to be able to view trials in Data Viewer; trial screens are not stored as single image files:

This is where it gets a bit more complicated. Presentation gives you flexibility as it is easy to combine picture elements during your experiment, for ex. generating screens containing several bitmaps and/or text, line drawings, etc. etc.. If this is what you want to do, the 'Screenshot Example' script probably comes closest to your design. However, using this design has important implications for timing. The problem is that Data Viewer requires a single image file to correctly view the trial, meaning that if the image does not exist as a single file, we have to present it first and then take a screenshot of the screen. Taking the screenshot is handled by Windows, and the time this takes may vary immensely, from a few ms to hundreds of ms. Some designs can work around this problem and some cannot, in which case the screens need to be pre-generated anyway.

The PresentTrialTakeScreenshotPresLink subroutine takes a screenshot right after the trial screen is presented. This presumes that it is ok if Presentation does nothing for (quite) a

bit of time after the screen comes on. Whether that is really ok depends on your design and how fast the computer is, i.e. how long it takes to make a screenshot (this time will vary quite a lot, even on a single PC). Note that during this period, data is still being recorded to the .edf-file. However, Presentation will not be able to process on-line tracking data (or do anything else) during this period. You may want to test how long taking a screenshot takes, as it may vary from a few ms to hundreds of ms, depending on the pc. (Though even on a single pc this duration is not fixed and depends on lots of things). Either way, to overcome timing problems, you can consider several solutions:

- If there is (ample) time at the beginning of the trial during which it is ok if Presentation cannot do anything: take screenshot right after presenting the screen, i.e. the way it is in the subroutine.

- If there is (ample) time to take a screenshot at the end of the trial by taking the screenshot after some crucial (response) period has happened: move the command which takes the screenshot to the end of the trial.

- Send a dummy image which shows the locations of the objects on screen in Data Viewer. This presumes that the Dummy Image can be generated beforehand, which is usually easy enough if the objects are always in the same position. You will not be able to see the actual images in Data Viewer, but you could deduce this from your Presentation logfile or extra information sent to the .edf-file.

- If none of the above is an option, it is best not to take a screenshot and pre-generate your trial screens after all, then use the PresentTrialPresLink function.

Will analyze trial data in a program other than Data Viewer:

It is possible to analyze the data in other programs, for ex. Matlab. This is not covered by this library, but perhaps you can take inspiration from the subroutines in this library. You should program your experiment to the specific requirements of your analysis program. The above considerations regarding trials and pre-generating images may or may not still apply.

Below follows a description of all the subroutines in this library and a few necessary global variables.

This library

`Global variables/objects`

bool bTrackingEnabled : bTrackingEnabled is used by all Eyetracking functions to determine whether Eyetracker-specific functionality should be used. Setting this to 'false' should enable you to run your experiment without being connected to the tracker. Do note that enabling/disabling this has a large effect and will cause some functions to nothing at all. Also, it will have an impact on timing as many of the tracking functions take quite some time to carry out, so always check your timing with the tracker connected as well!

eye_tracker tracker: 'tracker' will contain the eye_tracker object, which will be set to "PresLink" if bTracking is true. This assumes the PresLink Extension is activated in Presentation under the exact name "PresLink".

picture et_calibration: In addition to the global variables (which are placed in the library file itself), you also need to place a calibration picture object in your .sce-file, named **et_calibration**. This will enable you to set the background color for the calibration screen. Note that even though you create this object yourself in SDL, it should have the exact name et_calibration, as the extension assumes this name. The extension itself assumes this, not just this library, so do not modify the subroutines to use a different picture object, or the background will default to black.

`InitializePresLink(string sEdfFilename, bool bStartRecording)`

Description:

This subroutine carries out all of the tasks necessary to get the tracker ready for tracking and ensures the tracker uses the right settings.

- Check tracker version.
- Open .edf-file.
- Check screen width and height and communicate these to the tracker.
- Set which data is available through the link with the Eye Tacker, depending on the tracker version. You will often not need all the data which is initialized here. You can either choose to make the data available anyway, or remove irrelevant types from the string containing the data types.(It has not been tested whether this has any measurable effect on recording speed etc..)
- Start receiving position data for the left eye and the right eye. (alter this depending on your needs)
- Set calibration type
- Calibrate (gets you to the default tracker calibration routine, where you can use 'c' for calibration, 'v' for validation and 'o' to accept and continue.

- Do a drift correction
- If bStartRecording is set to **true**, start recording.

Parameters:

sEdfFilename: Name of the edf-file the eyetracker data will be recorded to. Make this dependent on subject name or number to make sure that old data is not overwritten. The argument should not have the .edf-extension; this is added by the subroutine.

bStartRecording: Whether to start recording right away after initialization or not. You can generally set this to **true** if you only need the on-line data (during the experiment) and do not need the .edf-file for later analysis in Data Viewer. In this case, you also do not need the subroutines below which are used to start and stop recording trials, you only need to use 'StopTrackingPresLink' when you want to stop recording and tracking . If you later want to analyze that data in Data Viewer, you probably want trials. In this case, set this variable to false, so recording will not be started in this subroutine. Instead, start recording at the beginning of each trial and end it at the end of the trial, using the appropriate subroutines explained below.

Warnings/notes:

You can also write a preamble to the .edf file, the code for this has been commented out because there is a known bug which occasionally causes crashes when writing the preamble. This does not happen on all PCs/trackers, so if you want a preamble, you can uncomment the line and check if it works.

```
StopTrackingPresLink(string sEdfFilename);
```

Description:

Ends recording and then ends tracking completely, closes edf-file and sends edf file to the Presentation logfile directory. Use this at the end of your experiment, or the end of a part where the tracker is used.

Parameters:

string sEdfFilename: filename of the.edf file you were recording to. Argument should not have the .edf extension, the subroutine appends this.

Warnings/notes:

If the experiment somehow crashes before this function can be executed, note that the .edf file will not be transferred to the Presentation PC, but should be stored on the tracker.

EndTrialPresLink

What the subroutine does:

Ends the current trial by stopping recording. Optionally, some lines can be uncommented to enable sending the TRIAL_RESULT to the tracker, if you need this for later analysis in Data Viewer. Without the commented lines, this subroutine mostly serves to make sure that if bTrackingEnabled == **false**, Presentation does not try to stop the tracker, which would otherwise result in errors if no tracker is connected.

PresentTrialTakeScreenshotPresLink(picture p_PresentThis, int iTrialID, string sGraphicFilename, string sDummyScreenPicture, int iPresentationTime)

Description:

Use this function if you want to present trial screens which have not been pre-generated. This applies to trial screens that do not use existing image files, or combine several existing images, and/or text objects. Because Data Viewer needs to work with a single image, this subroutine takes a screenshot after the trial is presented. If the trial uses a single picture which exists as a file (.bmp, .jpg, .png., etc.), there is no need to take a screenshot, and you should use a simpler subroutine than this one. It is recommended you read 'How to use this library' before using this subroutine.

Parameters:

p_PresentThis: the Presentation picture object you want to present

iTrialID: The Trial ID for the current trial (should be unique for each trial). Relevant for later analysis in Data Viewer, and ensures the screenshots get unique names.

sGraphicFilename: This subroutine takes a screenshot, which will get the name sGraphicFilename. The name should be specified without path or file extension. The Presentation logfile directory is automatically appended as the path, and .bmp as extension. The subroutine also adds the TrialID to the filename, to ensure that the filename is unique as long as iTrialID is unique for each trial (and as such the image files do not get overwritten).

sDummyScreenPicture: It is usually convenient to send an image of the trial to the screen of the EyetrackerPC, so the researcher can see the trial screen there, with the subject's gaze on it. Unfortunately, this requires a single image, which in this case does not exist before the trial is presented (that is, if you use this subroutine, it should not). We cannot send images after recording has started, so it is impossible to send the screenshot. To overcome this, we can send a dummy image which contains the outlines of the stimulus locations, so the researcher will still have a clue where the subjects gaze is with respect to stimulus objects. If you do not need this, it is not necessary to use sDummyScreenPicture, so you could remove the variable and any lines using it from the subroutine.

Step-by-step:

- Send some basic trial info to the .edf file. (TrialID etc).

- Send the name of the screenshot image file (which does not exist at this point!) to the .edf file, indicating that our screenshot of the trial screen will have this name and Data Viewer should display this image when viewing this trial.

- Send a dummy image containing placeholders for the image locations to the tracker, using 'transfer_image'. This gives us a clue where the subject is looking at. We cannot send the actual trial image, because it does not exist yet! You can also choose to send nothing, if you do not care or it is impractical to generate a dummy image beforehand, for ex. because image locations are random.

- Present the trial screen with Presentation.

- Take a screenshot. This may take a long time, during which the Presentation script does not continue.

Warnings/notes:

With tracking enabled, this subroutine requires a lot (hundreds of ms!) of preparation time before the trial is actually presented. If you want to accurately control the time of stimulus presentation, add a wait_until before the line: "p_PresentThis.present();". Make sure to wait until a time that leaves the subroutine more than enough time to finish all the preparations.

Note that taking a screenshot may take only a few ms, but may also take a lot of time (during which Presentation cannot do anything else), so be careful and check your timing! If you really need to grab tracker data or do something else as soon as possible after stimulus presentation, consider pre-generating your images and not using this subroutine.

Note that the script communicates the **image name** to the traker/.edf-file for later use in Data Viewer, using "!V IMGLOAD FILL ". This sends a name only and does not send the actual image, so the image does not have to exist yet, and in this case it does not, it will be generated later by taking a screenshot after the trial is presented. The **transfer_image** command does send an actual image, but merely serves to shows that image on the Eyelink PC, it will have no effect on the recorded data or on what you will later see in Data Viewer.

The **transfer_image** sends a single image to the Eyelink PC. This command can only be sent when not recording, meaning you cannot send images which have not been pre-generated. There is no way around this except not starting recording until after the trial has come on screen and a screenshot has been taken and sent to the tracker. This may take a lot of time and is generally not recommended.

`int CheckCoordinatesInArea(double d_X, double d_Y, array<int,2> ai_Locations)`

Description:

Checks if a point with coordinates **d_X** and **d_Y** is within (one of) the rectangular area(s) specified in ai_Locations. Returns the index of the first location which contains the point. Returns zero if no locations contain the point. Parameters are in double precision because Eyelink returns coordinates as doubles.

Parameters:

d_X and **d_Y**: x and y-coordinates of the point. Though these generally will be whole number pixel values, this parameter has double precision because Presentation returns eyepos-coordinates in double precision. The subroutine converts these doubles to integers, so be sure to use pixels and not your own custom units. The function returns the number of the area which contains the specified point, and returns zero if no locations contain the point.

ai_Locations: array which contains rectangular locations. Each area should be specified as [1] left X, [2] right X, [3] Bottom Y, [4] Top Y. For example, ai_Locations[3][4] will contain the TopY coordinate for area number three.

Warnings/notes:

This subroutine is intended for gaze coordinates but can of course be used to for ex. check if mouse coordinates are within a certain area. You might need to change the coordinate parameters from double to int for that.

Areas should not overlap, as the function ends as soon as it encounters an area which contains the point, so only the **first** area containing the point is returned.

MeasureGazeInArea(int iTargetArea, array<int,2> aiArea, int iTimeoutDuration)

Description:

Monitors gaze and checks if it is in a specific target area, by passing target area and aiArea to the subroutine *CheckCoordinatesInArea* (see above). If the gaze has been in the target area for a specific time (cumulatively), the function ends.

Parameters:

int iTargetArea: index of the target area within the array aiArea.

array<int,2> aiArea: Two dimensional array, containing one or more areas as specified by [1] left X, [2] right X, [3] Bottom Y, [4] Top Y (See *CheckCoordinatesInArea)*.

int iTimeoutDuration: The subroutine will end if the gaze has been in the target area for this duration.

Warnings/notes:

This subroutine shows the general principles of checking whether the gaze is in a certain area. It can of course be modified to have several target areas, or have another ending condition than the current timeout.

The subroutine currently uses measurements from the right eye. Depending on your wishes and the tracker you are using, this can easily be modified to track left_eye or (if available) average eye position instead.

If the gaze happens to be in the area for a single data point, this is not measured as 'time in area', because it is impossible to determine how long the gaze was in the area. Time is always measured between two data points.

This subroutine measures **cumulative** time in the target area, i.e. the time measurement does not reset if the gaze exists the area. It can easily be changed to require continuous time in area, i.e. reset when the gaze leaves the area.

The subroutine does not simply add up all the intervals between every two data points in the area. This would be much easier to understand, but will cause rounding errors. Instead, it measures the time the gaze enters the area and as long as the gaze stays in the area, updates the 'intermediate time in the area' by taking the time of the current data point minus the time the area was entered. This intermediate time is added to the 'total time in area' when the gaze leaves the area or the function ends. N.B.: when the gaze enters and leaves the area multiple times, this may still cause rounding errors, but much less than the alternative.

ShowGazeCoordinates(int iTargetArea, array<int,2> aiArea)

Description:

Dummy subroutine for testing purposes. Shows the gaze coordinates and the area the gaze is in until the user presses a button.

Parameters:

int iTargetArea: index of the target area within the array aiArea.

array<int,2> aiArea: Two dimensional array, containing one or more areas as specified by [1] left X, [2] right X, [3] Bottom Y, [4] Top Y (See *CheckCoordinatesInArea)*.

Warnings/notes:

This function assumes the existence of a picture object p_GazeCoordinates, and several text objects, which should be added in SDL (or changed to another object). See the .sce-file of the example experiment for this object.