ViewPoint EyeTracker Toolbox™ for MATLAB

a ***ViewPoint EyeTracker*** ® interface
to the ***MATLAB*** ® application
& to the ***Psychophysics Toolbox***

Apple Macintosh OSX & MS Windows

Software User Guide

Arrington Research

Verified the following on 2018-March-26

*ViewPoint -**_64.exe*	2.9.5.138
VPX_InterApp_64.dll	2.9.5.111
Tested with MATLAB:	'9.4.0.813654 (R2018a)'
Compilers	Microsoft Visual C++ 2015 (C) , or Microsoft Visual C++ 2015 (C)
ViewPointClient_32.exe	2.9.5.111
VPX_InterApp_32.dll	2.9.5.111
Tested with MATLAB:	8.1.0.604 (R2013a)
Compilers	Microsoft Visual C++ 2008 SP1 , or Lcc-win32 C 2.4.1

Arrington Research, Inc.
27237 N 71st Place, Scottsdale, AZ 85262
United States of America
www.ArringtonResearch.com
Phone +1-480-985-5810

ViewPoint-EyeTracker@ArringtonResearch.com

Path: X:\ARI-Shared\ARI Graphic Arts\ARI Documentation (Shared)\MATLAB\ViewPoint EyeTracker Toolbox - OO-012a - .odt
Date: 15:01:42 A12/P12, Dec 10, 2018

1. System requirements and supported versions

1.1. MAC

- MAC OSX 10.5 and newer
- MAC 32-bit needs MATLAB 7.5 (R2007b) and newer
- MAC 64-bit needs MATLAB 7.6 (R2008a) or newer.
- MAC 64-bit needs Perl available.
- MAC 64-bit needs compiler gcc/g++ 4.1.1 only.

MathWorks does not support `gcc/g++ 4.2.x`, as of this writing. Some newer versions of MATLAB will work with `4.2.x` but MathWorks does not document these versions since they do not test or support them. So if you are using `4.2.x` and the `LoadLibrary` function is failing, you may need to uninstall `4.2.x` and install `4.1.1`.

We have successfully run `gcc/g++ 4.2.1` with MATLAB R2009a, R2009b, and R2010a. We also have customers who have used it with MATLAB R2008a.

Some customers have installed the compiler from their OSX system disks and still had issues. You can use the commands `gcc -v` and `gcc --version` in a terminal window to verify the `gcc/g++` compiler is installed and what version it is. Some customers have failed to install the compiler from their system disks even though the install appeared to work; they finally got a compiler installed by installing XCode.

1.2. PC

- Windows 7 or newer.
- Windows 32-bit needs MATLAB 7.5 (R2007b) and newer.
- Windows 32-bit needs Microsoft Visual C++ 2008 SP1 , or Lcc-win32 C 2.4.1
- Windows 64-bit needs MATLAB 7.6 (R2008a) or newer.
- Windows 64-bit needs VC++ 2013, or 2015 (free versions is okay).
- Windows 64-bit needs Perl available.

2. Introduction

2.1. ViewPoint EyeTracker Toolbox

The documentation describes the *ViewPoint EyeTracker Toolbox ™* , an interface between the *ViewPoint EyeTracker*® (*ViewPoint*) and MATLAB®. It also provides an optional interface to the *Psychophysics Toolbox* for calibrating *ViewPoint* within that stimulus environment.

This documentation assumes that the user is familiar with *ViewPoint,* MATLAB, and optionally the *Psychophysics Toolbox*. The user should consult the respective manuals for information about how to use these products.

The *Viewpoint EyeTracker Toolbox* is a set of MATLAB `.m` files that provide functions to *(a)* access *ViewPoint* data, and *(b)* send Command Line Interface (**CLI**) instructions to *ViewPoint*.

2.2. Interface Overview

Viewpoint EyeTracker Toolbox provides functions for MATLAB that interface to the *ViewPoint* dynamic library. The dynamic library is called a ***DLL*** on *MS-Windows;* it's called a ***dylib*** on *Apple Macintosh, and it's called a Shared Object* (**.so**), *on Linux*.

The toolbox handles loading this dynamic library into MATLAB (using the `loadlibrary` function) and it wraps the function calls (made via the `calllib` function) in `.m` files to make them more user friendly.

You can also use the `loadlibrary` and the `calllib` functions directly if you prefer.

You should always disconnect from ViewPoint and unload the library every time you exit MATLAB.

2.3. Important Changes

MAC:

As of version `2.8.5`, the *Macintosh dylib* has the client built into it, which eliminates the separate *ViewPointClient* application. This allows layered applications and the *Viewpoint EyeTracker Toolbox* to directly create a network connection to the *ViewPoint EyeTracker* server. The following shows and example of how the MATLAB functions should be used:

```
vpx_ConnectToViewPoint( '192.168.1.99', 5000 );
vpx_DisconnectFromViewPoint();
```

PC:

The serial port interface program, RemoteLink™, is obsolete and is replaced by the Ethernet interface program ViewPointClient™.

2.4. Steps

To use the *Viewpoint EyeTracker Toolbox* follow the following steps. Here is an outline of the steps, each of these steps is detailed in *Section 3. Getting Started*.

1. Specify a compiler for MATLAB to use.
2. Locate and install the *Dynamic Library* file and the *Header* file.
3. Tell MATLAB where to find the *Viewpoint EyeTracker Toolbox* files.
4. Edit the **`vpx_Initialize.m`** file so that it species the correct file path for the **`vpxMATLAB.h`** file and the **`VPX_InterApp_64.dll`** file. (For Apple, use **`vpx.h`** if **`vpxMATLAB.h`** is not in your release.) *IMPORTANT: You MUST use the Dynamic Library that is connected to ViewPoint (if MATLAB is on the same machine) or the Dynamic Library that is connected to ViewPointClient; do not use other copies that may be in other folders.*
5. Start the *ViewPoint EyeTracker*. If using Ethernet to connect to a second computer, then start the *ViewPointClient* application on the second computer.
6. Call the toolbox function **`vpx_Initialize`** that initializes the toolbox and loads the library. (This step assumes you modified the **`vpx_Initialize.m`** file so that the file and folder paths are correct for your machine.)
7. Connect to *ViewPoint* (only if using Ethernet).
8. At this point you can access the live data from the *ViewPoint EyeTracker*. We recommend running the **`VPX_Matlab_Demo`** to verify things are working correctly.
9. Disconnect from *ViewPoint* (only if using Ethernet).
10. Call the toolbox function: **`vpx_Unload`** to unload the library.

3. Getting Started

3.1. Specify a compiler for MATLAB to use.

MAC:

On *Macintosh*, MATLAB typically uses the default compiler.

PC:

Enter the following:

```
mex -setup
```

A list of available compilers will be presented. Select one of either **Microsoft Visual C++ 2013 (C)** or **Microsoft Visual C++ 2015 (C)**. You may be asked to verify your selection. *The MinGW64 Compiler will NOT work with our DLL.*

Also verify that the compiler is compatible with the version of MATLAB that you are using; check the MathWorks website for details. For example:

```
https://www.mathworks.com/support/compilers.html
```

Example of using `mex -setup` in MATLAB *Command Window*

```
>> mex -setup
MEX configured to use 'Microsoft Visual C++ 2015 (C)' for C language compilation.
Warning: The MATLAB C and Fortran API has changed to support MATLAB
                 variables with more than 2^32-1 elements. You will be required
                 to update your code to utilize the new API.
                 You can find more information about this at:
                 https://www.mathworks.com/help/matlab/matlab_external/upgrading-mex-files-to-use-64-bit-api.html.

To choose a different C compiler, select one from the following:
MinGW64 Compiler (C)  mex -setup:'C:\Program Files\MATLAB\R2018a\bin\win64\mexopts\mingw64.xml' C
Microsoft Visual C++ 2013 (C)  mex -setup:'C:\Program Files\MATLAB\R2018a\bin\win64\mexopts\msvc2013.xml' C
Microsoft Visual C++ 2015 (C)  mex -setup:C:\Users\julie\AppData\Roaming\MathWorks\MATLAB\R2018a\mex_C_win64.xml C

To choose a different language, select one from the following:
 mex -setup C++
 mex -setup FORTRAN
Renamed options file 'C:\Users\julie\AppData\Roaming\MathWorks\MATLAB\R2018a\mex_C_win64.xml' to
'C:\Users\julie\AppData\Roaming\MathWorks\MATLAB\R2018a\mex_C_win64_backup.xml'.
MEX configured to use 'Microsoft Visual C++ 2013 (C)' for C language compilation.
Warning: The MATLAB C and Fortran API has changed to support MATLAB
                 variables with more than 2^32-1 elements. You will be required
                 to update your code to utilize the new API.
                 You can find more information about this at:
                 https://www.mathworks.com/help/matlab/matlab_external/upgrading-mex-files-to-use-64-bit-api.html.
>>
>> addpath('C:\ARI\DEV\MATLAB\ViewPoint_EyeTracker_Toolbox')
>> vpx_Initialize()
>> VPX_Matlab_Demo
```

3.2. Locate and install the *ViewPoint* library files

3.2.1. Locate and install the library files

MAC:

The administrator (super user) password will probably be required, because the dynamic library needs to be moved to a low-level part of the UNIX operating system. Open the **Terminal.app** from the *Utilities* folder inside the *Applications* folder. Copy the dynamic library using the following command:

```
sudo cp /InstallPath/ViewPoint/libvpx_interapp.dylib /usr/local/lib/
```

where ***InstallPath*** should be changed to the installation folder. Once the file is copied, you need to change the attributes on the file by entering the following command:

```
sudo chmod 755 /usr/local/lib/libvpx_interapp.dylib
```

PC:

The **VPX_InterApp_64.dll** file is already located in the **~/ViewPoint/** folder along with the *ViewPoint* executable file and should remain in this folder.
IMPORTANT: You MUST use the Dynamic Library that is connected to ViewPoint (if MATLAB is on the same machine) or the Dynamic Library that is connected to ViewPointClient; do not use other copies that may be in other folders.

You can verify which DLL that *ViewPoint* is using by checking: *ViewPoint* menu > **Help > Info > SysInfo** tab **> SDK & DLL** section > **DLL path:** line.

3.2.2. Locate and install the Header (*.h) file

Unlike the dynamic library file, the **.h** files can be placed most anywhere, however the location of these files must be specified when the *ViewPoint EyeTracker Toolbox* is initialized.

MAC:

When MATLAB is installed it creates a folder for user documents. This is a good place to put the *ViewPoint* files.

.../MATLAB/ViewPoint/

The **vpx.h** and **vpxtoolbox.h** files are located in the **~/ViewPoint/SDK/** folder.

PC:

Locate the **vpxMATLAB.h** file. This is just a text file and it does not matter where it is located. This file may be in various locations depending upon the version of *ViewPoint*, generally you can find it in one of the following locations:

.../ViewPoint/Interfaces/Programming/SDK/vpxMATLAB.h

.../ViewPoint/Interfaces/vpxMATLAB.h

.../ViewPoint/SDK/vpxMATLAB.h

3.3. Set the path to the toolbox

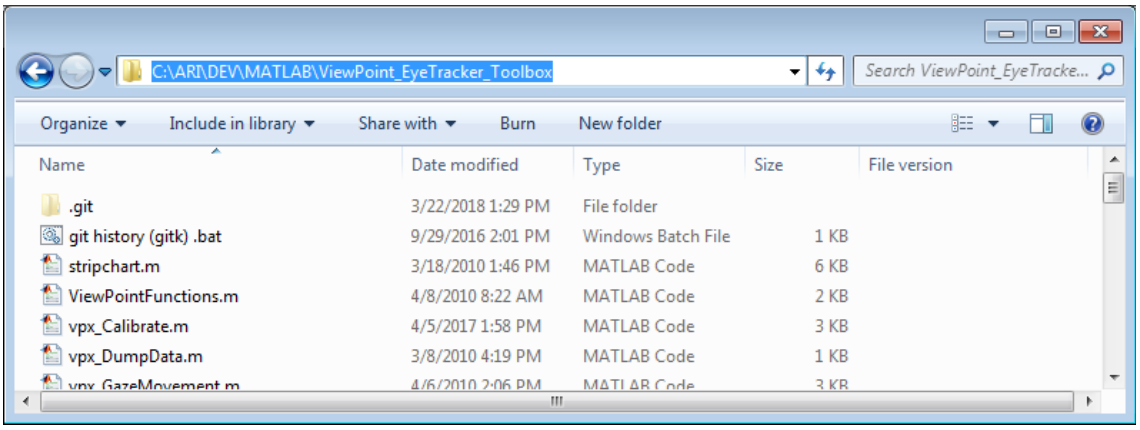
We must tell MATLAB where to find the *ViewPoint EyeTracker Toolbox ™*, that is, specify the file path to where the *Toolbox* resides. The default location has changed somewhat from one version to another, but in general it is somewhere under the **`…/ViewPoint/Interfaces/`** folder. The most recent version as of this writing has the folder at:

`...ViewPoint\Interfaces\3rdParty\Microsoft-Windows\MATLAB\ViewPoint_EyeTracker_Toolbox\`

You can specify this path using the MATLAB *Command Window* or using the MATLAB GUI interface; the latter has changed over different versions of MATLAB.

3.3.1. addpath function in MATLAB Command Window

The easiest way is to open the **`ViewPoint_EyeTracker_Toolbox`** folder, then click in the *Address Bar* (the editable text box that shows the path of the current folder) located at the top of the *Windows Explorer* window and copy the full path, as shown highlighted here:



and then paste it into the MATLAB *Command Window* as the argument for the **`addpath`** function. You should then see this path that you added in the list of paths that is displayed using the **`path`** function, as shown below.

Example of using `addpath` in MATLAB *Command Window*

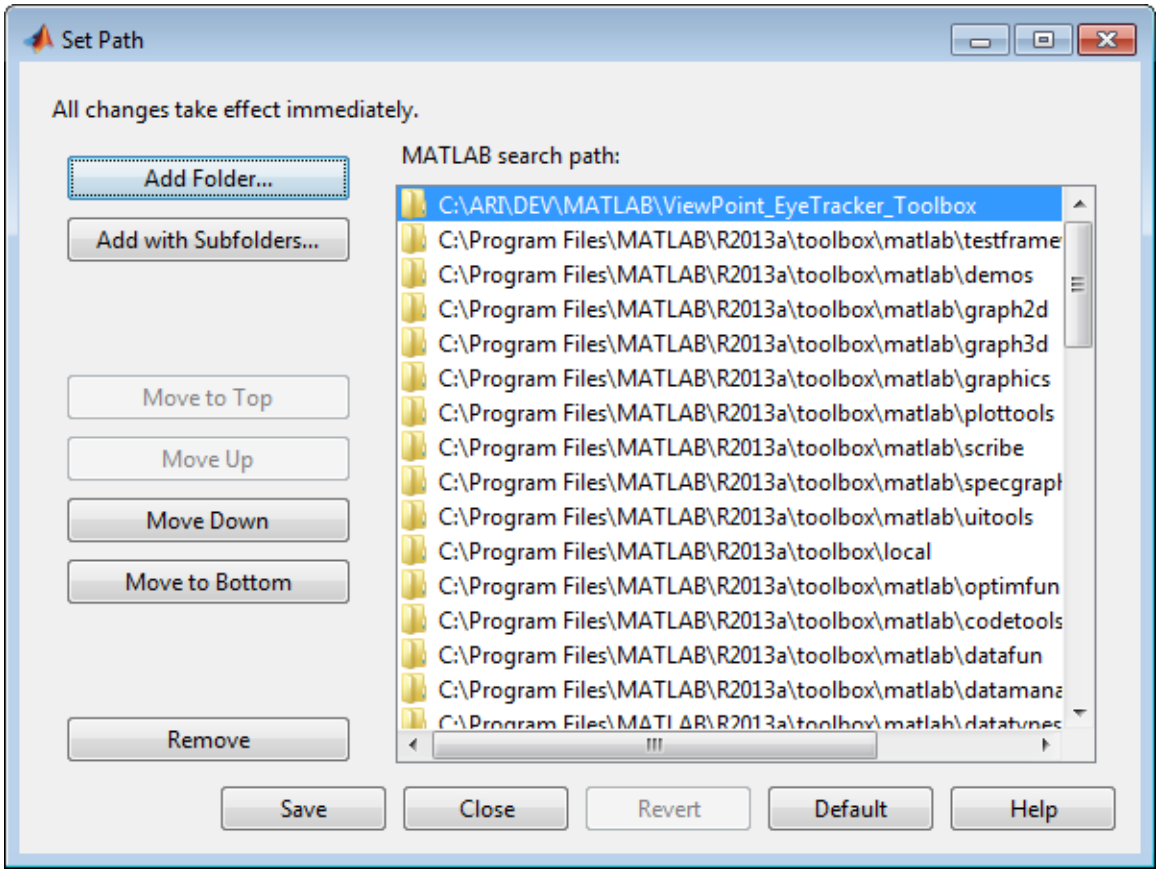
```
>> addpath('C:\ARI\DEV\MATLAB\ViewPoint_EyeTracker_Toolbox')
>> path

                MATLABPATH

     C:\ARI\DEV\MATLAB\ViewPoint_EyeTracker_Toolbox
     C:\Program Files\MATLAB\R2013a\toolbox\matlab\testframework
     C:\Program Files\MATLAB\R2013a\toolbox\matlab\demos
     C:\Program Files\MATLAB\R2013a\toolbox\matlab\graph2d
     .....
```

3.3.2. Using MATLAB GUI interface to Set Path

In some versions, in the MATLAB program > **HOME** tab, **ENVIRONMENT** group, select **Set Path.** This will bring up a window, as shown here below. Click [**Add Folder**] button and navigate to the desired folder. *Be sure to* [**Save**] when you are finished.



For other versions, in the MATLAB program, set the path by using the MATLAB menu: **File > Set Path > Add Folder ...**, browsing to the *ViewPoint_EyeTracker_Toolbox* folder, and clicking to add it. Finally, be sure to '**Save**' the settings, otherwise you will need to repeat this specification every time you restart MATLAB.

3.4. Edit the `vpx_Initialize.m` file

You **MUST** edit the `vpx_Initialize.m` file so that it specifies the correct file paths for the *Dynamic Library*, the *Toolbox* and the *Headings* files.

Mac:

```
vpxDylib = '/usr/local/lib/libvpx_interapp.dylib'
vpxToolbox = '/Users/YourName/Documents/MATLAB/ViewPoint/vpxtoolbox.h'
vpxH = '/Users/YourName/Documents/MATLAB/ViewPoint/vpx.h'
vpx_initialize( vpxDylib, vpxToolbox, vpxH )
```

PC (newer versions):

```
vpxDL      = 'C:\ARI\ViewPoint\VPX_InterApp_64.dll';
vpxDH      = 'C:\ARI\ViewPoint\Interfaces\vpxMATLAB.h';
```

The easiest way is to open the .../`ViewPoint/` folder, then click in the *Address Bar* (the editable text box that shows the path of the current folder) located at the top of the *Windows Explorer* window, copy the full path, and paste it into the file.

Important:

- *Must use BACK-SLASHES on Microsoft Windows.*
- *Must use SINGLE-QUOTES.*

PC (older versions):

Specifies PATH for the `vpx.h` and `vpxToolbox.h` files.

3.5. Start *ViewPoint* or *ViewPointClient*

PC:

If using Ethernet, you should now start the *ViewPointClient* application; you can connect *ViewPointClient* to the *ViewPoint* server later.

If connecting directly to *ViewPoint*, you should start *ViewPoint* now.

Always do this before initializing the ViewPoint EyeTracker Toolbox.

3.6. Initialize the *ViewPoint EyeTracker Toolbox*

For the *Toolbox* to work, the file `vpx_Initialize.m` must have the correct file paths specified for the *Dynamically Linked Library* and *Header* files. Editing the file `vpx_Initialize.m` should only need to be done once.

In MATLAB run the `vpx_Initialize` function (*without the* `.m` *at the end*). This loads the dynamic library. See the *Individual Function Descriptions* section for more information.

MAC:

On the Mac, `vpx_Initialize` also calls `VPX_so_init` which initializes *dylib* shared memory.

3.7. Connect to *ViewPoint*

3.7.1. MATLAB is on same machine and using same DLL as *ViewPoint* is using.

Make sure that both MATLAB and *ViewPoint* are actually using the exact same DLL. The most frequent problem is when MATLAB is loading a different copy of the DLL than the one that *ViewPoint* is using. *This mistake can happen when someone has copied the DLL to the MATLAB folder, or has installed it in the root, or is using another copy that exists somewhere in the ViewPoint subdirectories, such as the one in the SDK folder.* If MATLAB has loaded a different copy of the DLL, no communication with *ViewPoint* can occur via the DLL and the data will all be zeros.

3.7.2. MATLAB and *ViewPoint* are on different computers connected by Ethernet.

Skip this step if *ViewPoint* and MATLAB are on the same computer and will be using the same dynamic library. If using Ethernet a connection to *ViewPoint* must be established.

Make sure *ViewPoint* is running on the remote system. The default port number is 5000 and this will rarely need to be changed, only if there is a port conflict on that port number, which rarely happens. You will need to determine the IP address on the computer running *ViewPoint*. This can be found in the *ViewPoint* Status window; it will be displayed something like this:

```
Link: Ethernet Server: 192.168.1.7, Port: 5000
```

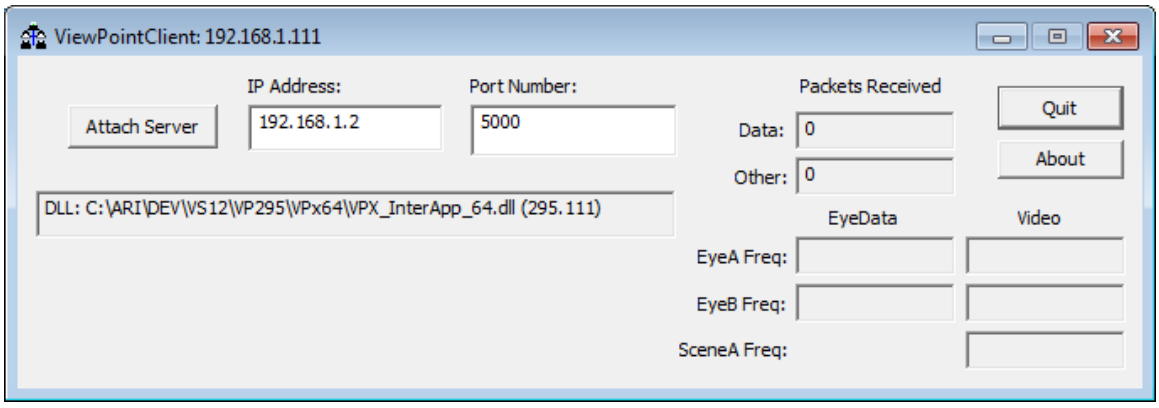
To connect to *ViewPoint* follow the below steps:

MAC: In the MATLAB *Workspace* window, enter the following:

```
vpx_ConnectToViewPoint('192.168.1.2',5000)
```

As of version 2.8.5, the Macintosh dylib has the client built into it, which eliminates the separate ViewPointClient application. This allows layered applications and the Viewpoint EyeTracker Toolbox to directly create a network connection to ViewPoint.

PC: Run the *ViewPointClient* application. You can verify which DLL that *ViewPointClient* is using by examining the text box when the application is first launched.



Next, specify the *ViewPoint EyeTracker IP-Address* and *Port-Number*, then press button [**Attach Server**].

You should no longer use the old serial port interface program, RemoteLink™; it is now obsolete and is replaced by the Ethernet interface program ViewPointClient™.

Once connected, the demonstration program that is described next will provide a gratifying proof that all is working correctly. *Note carefully that the demonstration program must be stopped before proceeding to make individual functions calls.*

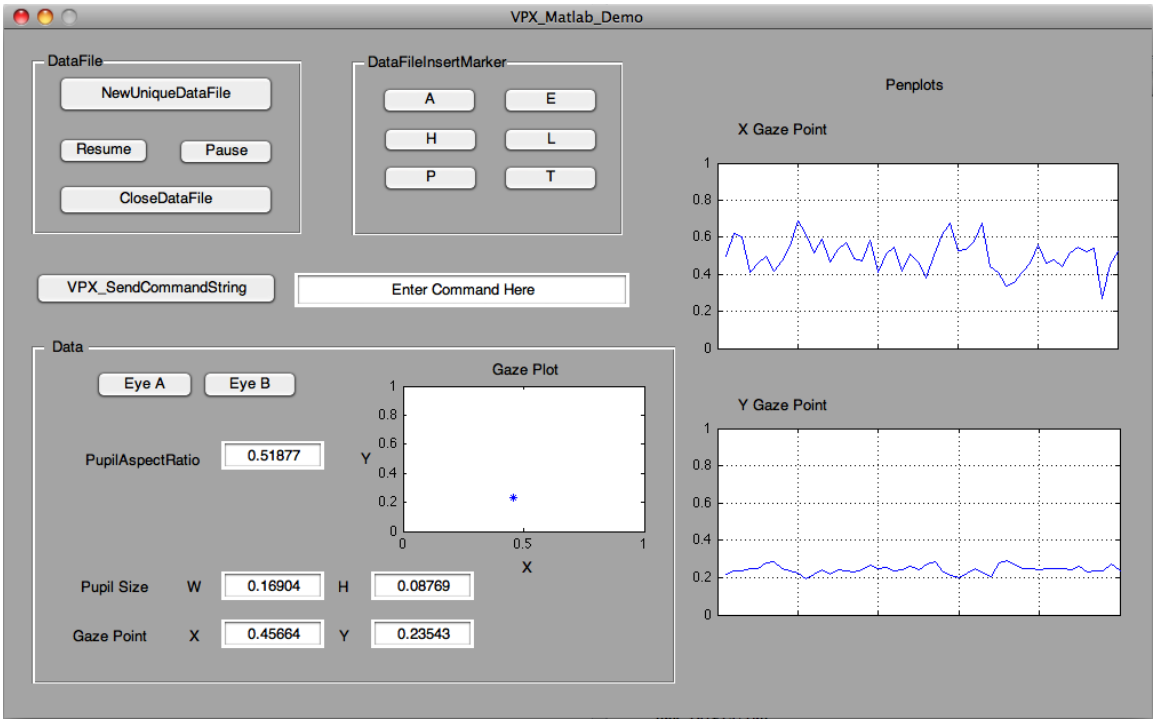
3.8. Using *VPX_Matlab_Demo*

We have included a demonstration interface that provides buttons for opening, pausing, resuming and closing *ViewPoint* data files, manually inserting synchronization markers, sending command strings to *ViewPoint*, and getting "real-time" data from *ViewPoint*. We suggest that you try this first – it may provide all the functionality that you require. To start the demo do the following:

At the MATLAB command prompt in the MATLAB *Workspace* window, type:

```
vpx_Matlab_Demo
```

The following user interface will be displayed.



Note carefully that the demonstration program must be stopped before proceeding to make individual functions calls.

If `vpx_Initialize` has not been done, then the following error occurs:

```
>> VPX_Matlab_Demo
Error using feval
. . .
```

To correct this you will now need to first do `vpx_Unload` before `vpx_Initialize` will succeed:

```
>> vpx_Unload
>> vpx_Initialize
>> VPX_Matlab_Demo
```

3.9. Disconnect from *ViewPoint*

If *ViewPoint* is the distributor then this step is not required since *ViewPoint* and MATLAB will be using the same dynamic library on the same system. If *ViewPoint* is not the distributor then a connection to *ViewPoint* was required and must be disconnected before closing MATLAB.

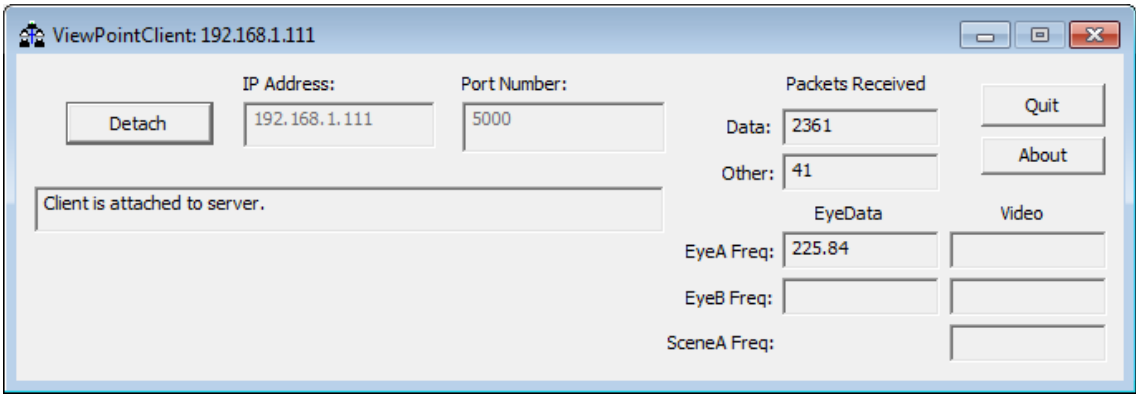
To disconnect from *ViewPoint* follow the below steps:

MAC: In the MATLAB *Workspace* window, enter the following:

```
vpx_DisconnectFromViewPoint
```

PC: In the *ViewPointClient* application, press button [**Detach**].

Failure to perform this step may produce undesired results in MATLAB.



3.10. Unload the library.

The last step that is always required before closing MATLAB is to unload the library. In the MATLAB *Workspace* window, enter the following:

```
vpx_Unload
```

Once the library is unloaded, you can close MATLAB. Failure to perform this step may produce undesired results in MATLAB.

4. *ViewPoint EyeTracker Toolbox* functions

4.1. Functions List and Online Help

The *ViewPoint Toolbox* provides a rich set of functions that can be used just like MATLAB functions. You can obtain a complete list of the *ViewPoint Toolbox* functions by entering the following in the MATLAB *Workspace*:

```
ViewPointFunctions
```

Help for each *ViewPoint EyeTracker Toolbox* function can be obtained online by entering the following in the MATLAB *Workspace*, at the command line prompt:

```
help functionName
```

where `functionName` is the name of the function that you want help with.

The next section describes each function in detail.

4.2. Optional Arguments

Many of the functions are overloaded with optional arguments.

Most of the functions that return eye data can be called with an optional argument to specify the eye: pass `0` to indicate `EYE_A`, or pass `1` to indicate `EYE_B`. The function will default to `EYE_A` if it is called without any arguments. If you specify `EYE_B`, *make sure that the ViewPoint EyeTracker is running in* binocular *mode*. For example, all of the following are equivalent:

```
[x,y] = vpx_GetGazePointSmoothed(0)
[x,y] = vpx_GetGazePointSmoothed()
[x,y] = vpx_GetGazePointSmoothed
```

using no argument will default to `EYE_A`, whereas

```
[x,y] = vpx_GetGazePointSmoothed(1)
```

gets the (x,y) calculated smoothed position of gaze (POG) for `EYE_B`.

4.3. Additional Functions

The *ViewPoint EyeTracker Toolbox* provides all the basic functions that most users need. Sometimes newer SDK functions may not have *Toolbox* functions provided yet through `.m` files, but these can be easily added by following the templates of the provided `.m` functions.

Alternatively, you do not really need `.m` wrappers at all to call the SDK functions; you can call them directly with `calllib`, just as is done in the `.m` file.

The following sections describe most of the Pre-Built function in the *Toolbox*.

If a particular function or some of its functionality does not yet appear in the Mac distribution, consult the `.m` file in the PC distribution.

4.4. MATLAB syntax and Return Values

The following calls to this scalar, *One Output*, function are all equivalent:

```
>> velocity = vpx_GetTotalVelocity
velocity =
    0.0115
>> [velocity] = vpx_GetTotalVelocity
velocity =
    0.0115
```

However, functions with *Multiple Outputs* should use square brackets containing the returned outputs, for example:

```
>> [dx,dy] = vpx_GetComponentVelocity
dx =
  0.0100
dy =
  0.0100
```

The value may be incorrect if the square brackets are not used with Multiple Outputs.

4.5. Pre-Built functions

4.5.1. Initialization

```
vpx_Initialize
vpx_Initialize( vpxDylib, vpxToolbox, vpxH )  (MAC)
vpx_Initialize( vpxDL, vpxH )  (PC)
```

This function must be called before any other *ViewPoint EyeTracker Toolbox* function can be used. This initializes the toolbox by loading the *dynamic library* and associated `.h` files. The user can simply call `vpx_Initialize` if the `.m` file is edited to specify the file paths, otherwise they can be specified as arguments.

When exiting MATLAB you must always unload the library by calling `vpx_Unload`.

MAC:

If you decide not to use vpx_Initialize and use `loadlibrary` manually, make sure that you also do: `calllib('vpx','VPX_so_init')`. *This must be called before any data in the dynamic library is accessed, otherwise MATLAB will likely terminate!*

MAC:

```
vpxDylib   = '/usr/local/lib/libvpx_interapp.dylib'
vpxToolbox = '/InstallPath/ViewPoint/SDK/vpxtoolbox.h'
vpxH       = '/InstallPath/ViewPoint/SDK/vpx.h'
vpx_Initialize( vpxDylib, vpxToolbox, vpxH )
```

where `InstallPath` needs to be changed to the installation path of *ViewPoint*.

PC:

```
vpxDL     = '\InstallPath\ViewPoint\VPX_InterApp_64.dll'
vpxMH     = '\InstallPath\ViewPoint\Interfaces\vpxMATLAB.h'
vpx_Initialize( vpxDL, vpxMH )
```

where `InstallPath` needs to be changed to the installation path of the `…\ViewPoint\` folder. For the PC version, make sure to use '\' and NOT '/' in the paths. Case sensitivity may be an issue with some MATLAB versions so always use the exact case of your paths and files.

It is recommended that you edit the `vpx_Initialize.m` file and change the paths so that you can call `vpx_Initialize` without parameters every time you run MATLAB and don't have to re-specify the variable paths each time.

4.5.2. Unload

```
vpx_Unload
```

This function unloads the *ViewPoint* dynamic library that was loaded by **vpx_Initialize**. It does not unload the *ViewPoint EyeTracker Toolbox*. Most of the *Toolbox* functions will not run after **vpx_Unload**. To use the functions one has to initialize *ViewPoint* by calling **vpx_Initialize**.

4.5.3. ConnectToViewPoint (**MAC only**)

```
r = vpx_ConnectToViewPoint( ipAddressDotQuadString, portNumber)
```

This uses the specified *IP-Address* and *Port-Number* to connect the Ethernet client (that is built into the *dylib)* to the server (that is built into *ViewPoint)*. *This function is only available for the MAC*, since the client socket is built into the *dylib*.

```
Return values:
        1 - already connected
        0 - connect success
       -1 - connect error
       -2 - negotiation send error
```

When exiting MATLAB you must always disconnect from ViewPoint by calling **vpx_DisconnectFromViewPoint**

Example:

```
result = vpx_ConnectToViewPoint( '192.168.1.2', 5000 )
```

4.5.4. Disconnectedness (**MAC only**)

```
result = vpx_DisconnectFromViewPoint()
```

Disconnects the Ethernet client that is built into the *dylib* from the server that is built into *ViewPoint*. *This function is only available for the MAC*, since the client socket code is built into the *dylib*.

```
Return values:
        1 - already disconnected
        0 - disconnect success
       -1 - disconnect error
```

When exiting MATLAB you must always disconnect from ViewPoint before calling **vpx_Unload**.

4.5.5. SendCommandString

```
vpx_SendCommandString( str )
```

Sends a command string to the *ViewPoint EyeTracker*. MATLAB can completely control the *ViewPoint EyeTracker* by sending *Command Line Interface* (**CLI**) instructions. Every graphical user interface (GUI) item (menu item, button, etc.) in *ViewPoint* has an equivalent CLI instruction.

See the *ViewPoint EyeTracker Manual* for a description of the syntax and list of commands.

MATLAB strings are enclosed within single quotes, but make sure that the text strings within the CLI command strings are enclosed in double quotes.

Example:

```
vpx_SendCommandString(' dataFile_NewName "0059.txt" ')
vpx_SendCommandString(' dataFile_InsertMarker K; dataFile_Pause ')
vpx_SendCommandString(' say "Hi from MATLAB"  ')
```

4.5.6. GetGaze*

```
[x,y] = vpx_GetGazePoint
[x,y] = vpx_GetGazePoint( eyeID )
[x,y] = vpx_GetGazePointSmoothed( )
[x,y] = vpx_GetGazePointSmoothed( eyeID )
[x,y] = vpx_GetGazePointCorrected
[x,y] = vpx_GetGazePointCorrected( eyeID )
[x,y] = vpx_GetGazeBinocular
```

Retrieves the calculated position of gaze.

`vpx_GetGazePoint` returns the pure POG,

`vpx_GetGazePointSmoothed` returns the smoothed POG.

`vpx_GetGazePointCorrected` returns the corrected POG that may include smoothing, averaging, parallax correction, nudging, etc… In general, it is best to use this.

`vpx_GetGazeBinocular` returns Corrected Eye_A values if monocular, or Corrected Binocular Averaged.

See also: `vpx_GetGazeAngle, vpx_GetPupilPoint, vpx_GetGlintPoint`

4.5.7. GetGazeAngle

```
[azimuth, elevation] = vpx_GetGazeAngle
[azimuth, elevation] = vpx_GetGazeAngle( eyeID )
[azimuth, elevation] = vpx_GetGazeAngleSmoothed
[azimuth, elevation] = vpx_GetGazeAngleSmoothed( eyeID )
```

Retrieves the calculated component angles of gaze.

Note: The angles are from trigonometric calculations based on the values that the user has measured and set for the window size (horizontal & vertical) and the viewing distance. Refer to section 4.6.3 on `vpx_GeometryGrid`.

This is NOT for use with the head tracker option.

4.5.8. GetFixationSeconds

```
fixationSec = vpx_GetFixationSeconds
fixationSec = vpx_GetFixationSeconds( eyeID )
```

Retrieves the number of seconds that the total velocity has been below the *VelocityCriterion*.

A zero value indicates a saccade is occurring.

See also: `vpx_GetTotalVelocity`

4.5.9. GetTotalVelocity

```
velocity = vpx_GetTotalVelocity
velocity = vpx_GetTotalVelocity( eyeID )
```

Retrieves the total velocity of movement in the (x,y) plane. That is, the first derivative of the (smoothed) position of gaze.

See also: `vpx_GetComponentVelocity, vpx_GetFixationSeconds`

4.5.10. GetComponentVelocity

```
[dx,dy] = vpx_GetComponentVelocity
[dx,dy] = vpx_GetComponentVelocity( eyeID )
```

Retrieves the x- and y-components of the eye movement velocity.

See also: `vpx_GetTotalVelocity`

4.5.11. GetPupilSize

```
[ majorAxis, minorAxis ] = vpx_GetPupilSize
[ pupilWidth, pupilHeight ] = vpx_GetPupilSize( eyeID )
```

Retrieves the raw size (width and height) of the oval or elliptical fit to the pupil. When using ellipse fitting, the width is always the major axis, regardless of rotation angle. When using the old oval fit (unrotated ellipse) method, the width is the horizontal width of the bounding box.

NOTE: the returned values are commensurable, such that aspect = x / y, they are both normalized with respect to the horizontal width of the *EyeCamera* window (previous versions normalized the x and y component separately and presumed a 4:3 aspect ratio that required y*0.75).

See also: `vpx_GetPupilAspectRatio`

4.5.12. GetPupilAspectRatio

```
aspect = vpx_GetPupilAspectRatio
aspect = vpx_GetPupilAspectRatio( eyeID )
```

Retrieves the dimensionless value of pupil aspect ratio. The ratio is independent of the *EyeCamera* window shape. Note: a circular pupil will produce a value of 1.0.

See also: `vpx_GetPupilSize`

4.5.13. GetPupilPoint

```
[x,y] = vpx_GetPupilPoint
[x,y] = vpx_GetPupilPoint( eyeID )
```

Retrieves the raw normalized (x,y) location of the center of the pupil in the *EyeSpace*. This is the center of the elliptical fit or oval fit to the pupil, or the centroid, depending upon the method selected.

4.5.14. GetGlintPoint

```
[x,y] = vpx_GetGlintPoint
[x,y] = vpx_GetGlintPoint( eyeID )
```

Retrieves the raw normalized (x,y) location of the center of the glint in the *EyeSpace*. This is the center of the oval fit to the glint or the centroid, depending upon the method selected.

4.5.15. GetDiffVector

```
[dx,dy] = vpx_GetDiffVector
[dx,dy] = vpx_GetDiffVector( eyeID )
```

Retrieves the raw normalized vector difference between the centers of the pupil and the glint in the *EyeSpace*.

4.5.16. GetTorsion

```
t = vpx_GetTorsion
t = vpx_GetTorsion( eyeID )
```

Retrieves torsion in degrees. *The ViewPoint torsion measurement must be turned on*; it is off by default. Requires the *Torsion Option*.

4.5.17. GetDataQuality

```
q = vpx_GetDataQuality
q = vpx_GetDataQuality( eyeID )
```

Retrieves the quality code for the eye data. See **VPX.h** for a list of data quality constants. Returns an **int32** value.

4.5.18. GetDataTime

```
t = vpx_GetDataTime
t = vpx_GetDataTime( eyeID )
```

Retrieves the time (double floating point, high precision time, in seconds) that the video frame became available for the current data point, before video image processing and other calculations were done.

See also: **vpx_GetDataDeltaTime**

4.5.19. GetDataDeltaTime

```
dt = vpx_GetDataDeltaTime
dt = vpx_GetDataDeltaTime( eyeID )
```

Retrieves the time interval (double floating point, high precision time interval, in seconds) between the last two *dataTime* values.

4.5.20. GetROI_RealRect

```
rr = vpx_GetROI_RealRect( roiN )
```

Retrieves the normalized floating point coordinates for the specified region of interest (ROI), aka, area of interest (AOI), aka window discriminator box. This function returns *One Output*, a structure.

Example:

```
rr = vpx_GetROI_RealRect(9)
rr =
  struct with fields:
      left: 0.4400
       top: 0.0200
     right: 0.5600
    bottom: 0.1800
>> rr.top
ans =
    0.0200
```

4.5.21. ROI_GetHitListItem

```
roiID = vpx_ROI_GetHitListItem( eyeID, NthHit )
```

The ROI may be overlapped, such that the gaze point may be in more than one ROI at the same time. This function returns the ROI index number of the Nth ROI that the gaze point is in. Note that this is calculated separately for each eye. The ***NthHit*** argument starts at zero. The functions may be called repeatedly to obtain successive ROI index values until no more ROI are in the hit list. If the ***NthHit*** argument is greater than the hit count, then the function returns the value ROI_NOT_HIT = -999. The test is for values inside the ROI box values, not resting on the box lines.

Monocular *ViewPoint* users must specify **Eye_A**.

Example:

```
roiID = vpx_ROI_GetHitListItem( 0, 0 )
// this gets the first (lowest numbered) ROI hit by EYE_A
roiID = vpx_ROI_GetHitListItem( 1, 1 )
// this gets the second ROI hit by EYE_B
```

See also: **vpx_ROI_GetHitListLength**

4.5.22. ROI_GetHitListLength

```
len = vpx_ROI_GetHitListLength( eyeID )
```

The ROI may be overlapped, so a gaze point may be in more than one ROI.

This function returns a count of the number of ROI the gaze point is in. This is calculated for each eye since the gaze point may be different for the left and right eyes. If the gaze point is not in any ROI, a zero value is returned. If the gaze point was in three overlapping ROI, the value three is returned, etc.

This particular function always requires an argument, so monocular ViewPoint users must specify `0` *for* `Eye_A` *for this function, it will not use a default, as most other functions do.*

Example:

```
>> len = vpx_ROI_GetHitListLength(0)
```

See also: `vpx_ROI_GetHitListItem`

4.5.23. ROI_MakeHitListString (PC only)

```
[liststr] = vpx_ROI_MakeHitListString( eyeID, maxStringLength, indicateOverflow, noHitsString )
```

The string lists the ROI that were hit, eg: "2,45,88". If (*indicateOverflow* == true) then a "+" at the end of the string will be used to indicate that there were additional ROI that could not fit in the given string.

Makes a string with at most *maxStringLength* characters. Specifying a *maxStringLength* of 2 will effectively limit the reported ROI to the first. For double digit numbers, prints both digits if possible, otherwise prints nothing. Does not leave a dangling comma separator.

If there are no ROI hit, then the *noHitsString* is returned, e.g.: " -None- "

The return value is the length of the string.

Example:

```
>> vpx_ROI_MakeHitListString(0,20,true,'-NoneHit')
HitRegions =
    '1'
>> vpx_ROI_MakeHitListString(1,20,true,'-NoneHit')
HitRegions =
    '-NoneHit'
```

4.5.24. GetCalibrationStimulusPoint

```
[x,y] = vpx_GetCalibrationStimulusPoint( ixPt )
[x,y] = vpx_GetCalibrationStimulusPoint( ixPt, eyeID )
```

Returns (x,y) coordinate in normalized (0.0 .. 1.0) form of the calibration point specified by the index argument in the range [1 .. 72].

Note that the optional ***eyeID*** is the second argument.

Example:

```
>> [x,y] = vpx_GetCalibrationStimulusPoint(3)
x =
    0.1000
y =
    0.6333
```

4.5.25. GetHeadPositionAngle

```
[pa] = vpx_GetHeadPositionAngle
```

Retrieves head position and angle data. The **PositionAngle** structure is defined in **VPX.h**. This function returns *One Output*, a structure with 6 elements.

*** available with head tracker option only ***

NOTE: Subject to change.

Example:

```
>> [pa] = vpx_GetHeadPositionAngle
pa =
  struct with fields:
        x: 0
        y: 0
        z: 0
     roll: 0
    pitch: 0
      yaw: 0
>> pa.roll
ans =
     0
```

4.5.26. Trademark

vpx_Trademark

This function displays the trademark information for the *ViewPoint EyeTracker Toolbox™*.

Example:

```
>> vpx_Trademark
-----------------------------------------------------------------
ViewPoint EyeTracker Toolbox (TM)
Copyright 2005-2018, Arrington Research, Inc.
All rights reserved.
-----------------------------------------------------------------
```

4.5.27. GetStatus

[n] = vpx_GetStatus(*statusIndex*)

Returns the current status various things in *ViewPoint* or the library client. The integer ***statusIndex*** can be any of the following:

1	VPX_STATUS_ViewPointIsRunning
2	VPX_STATUS_VideoIsFrozen
3	VPX_STATUS_DataFileIsOpen
4	VPX_STATUS_DataFileIsPaused
5	VPX_STATUS_AutoThresholdInProgress
6	VPX_STATUS_CalibrationInProgress
7	VPX_STATUS_StimulusImageShape
8	VPX_STATUS_BinocularModeActive
9	VPX_STATUS_SceneVideoActive
10	VPX_STATUS_DistributorAttached
11	VPX_STATUS_CalibrationPoints
12	VPX_STATUS_TTL_InValues
13	VPX_STATUS_TTL_OutValues
14	VPX_STATUS_TorsionActive
15	VPX_STATUS_BinocularAveraging

Check the latest *ViewPoint UserGuide* for a complete list as well as a description of the meanings of the return values, some of which are more than just boolean.

4.6. Psychophysics Toolbox Functions

You can download the *Psychophysics Toolbox* from

`http://www.psychtoolbox.org` and install it as follows:

Set path by using the MATLAB menu: **File > Set Path > Add Folder ...**, browsing to the *Psychophysics Toolbox* folder, and clicking to add it. Finally, be sure to '**Save**' the setting.

4.6.1. Calibrate

```
vpx_Calibrate()
vpx_Calibrate( nPoints )
```

This function performs calibration, by communicating with the *ViewPoint EyeTracker*. The calibration snap mode is on and the calibration stimulus points are presented on the screen. The number of calibration points can be passed as an argument (`nPoints`). If no argument is passed, `12` calibration points are used. You can re-present a single stimulus point by specifying the negative of that stimulus point number; for example, `-3` will re-presents only stimulus point number **3**.

4.6.2. GazeMovement

```
vpx_GazeMovement
vpx_GazeMovement( roiCheckBool )
```

This is a demonstration that samples the position of gaze every second and displays a white square on the screen such that it moves on the screen with the eye.

Optionally, if the we pass a value of `1` (`roiCheckBool`=true) the demo will be set to check whether the gaze point is within a region of interest (ROI) and change the color of the square (to green) if it is inside an ROI.

Note 1: Currently only monocular.

Note 2: Accurate position of gaze requires that calibration has been performed.

To halt the program:

Bring forward MATLAB command window (`Alt-Tab`) and press `Ctrl-C` (type a "`c`" while holding down the "`Ctrl`" key.). This halts program.

Mac OS:

`Option + Command + Escape` will close down MATLAB.

4.6.3. Geometry Grid

vpx_GeometryGrid

ViewPoint provides an option for getting the gaze angle (c.f. **vpx_getGazeAngle**) These angles are from trigonometric calculations based on the values that the user has measured and set for the window size (horizontal & vertical) and the viewing distance.

vpx_GeometryGrid displays two thick white lines, one horizontal and one vertical in the *Psychophysics Toolbox* window. The horizontal and vertical line lengths and viewing distance between the center of the screen and the subjects eye can be entered by adjusting the sliders in the *Viewpoint* **Geometry** window > **2D** tab.

The adjustment can be done as follows:

Select *ViewPoint* menu: **Stimuli > Geometry setup …** . The *Geometry* setup window will be displayed, the sliders can be adjusted to indicate:

(a) Viewing distance from the subjects eye to the center of the display screen.

(b) Length of horizontal and vertical lines.

The user can measure the horizontal and the vertical line. For the units of measurement please check the *Geometry* window.

To halt the program:

Bring forward MATLAB command window (**Alt-Tab**) and press **Ctrl-C** (type a "**c**" while holding down the "**Ctrl**" key.). This halts program.

Mac OS:

Option + Command + Escape will close down MATLAB.

5. Other MATLAB commands

The following MATLAB calls should not be necessary in normal use, but are provided to help if problems arise.

5.1. rehash

`rehash toolbox` performs the same updates as **`rehash path`**, except it updates the list of known files and classes for all directories on the search path, including those in matlabroot/toolbox. Run **`rehash toolbox`** *when you change, add, or remove files in matlabroot/toolbox during a session*. Typically, you should not make changes to files and directories in matlabroot/toolbox. Use **`help rehash`** for details.

5.2. library routines

```
libfunctionsview('vpx')
libfunctions('vpx') (newer version)
   shows all functions and their arguments in the library
libisloaded('vpx')
   returns 1 if loaded, otherwise 0
```

6. Trouble Shooting

6.1. PC file path names require Back-Slashes

On a Windows PC, *you must use BACK-SLASH in file path names*, otherwise you will get the following error:

```
Error using loadlibrary (line 419)
There was an error loading the library.
[ the full path dll file name that you specified ]
The specified module could not be found.
```

6.2. Thunk errors

*`thunk_pcwin64.dll` exp

A thunk file is a compatibility layer to a 64-bit library generated by MATLAB. In principle this is easy: just i*nstall the appropriate compiler* and let MATLAB generate the correct thunk file.

If you have problems, you are *probably missing the compiler run-time libraries* for the compiler you used to build the thunk file.

< `http://www.mathworks.com/help/matlab/ref/loadlibrary.html` >

7. Hints

7.1. Launching *ViewPoint* from MATLAB on a PC

```
hfile = 'C:\ViewPoint\Interfaces\vpxMATLAB.h';
dlib  = 'C:\ViewPoint\VPX_InterApp_64.dll'
[notfound,warnings]=loadlibrary( dlib, hfile );
calllib('VPX_InterApp_64','VPX_LaunchApp','C:\ViewPoint\ViewPoint_64.exe',' ');
```

7.2. 32-bit MATLAB

It is not possible to use 32-bit MATLAB with 64-bit *ViewPoint* directly, because the **VPX_InterApp_32.dll** and **VPX_InterApp_64.dll** files are incompatible.

You can however use 32-bit MATLAB with **VPX_InterApp_32.dll** shared with a the 32-bit version of the client, **ViewPointClient_32.exe**, that then communicates via Ethernet with the 64-bit version of *ViewPoint*.

8. Document Format: Heading 1 (Calibri Reg. 18, Blue 5)

8.1. Styles: Heading 2 (Calibri Regular 16, Cyan-7)

8.1.1. Paragraph Styles: Heading 3 (Arial Regular 14, User Teal)

Text body (Calibri regular 12)

Text body (Calibri regular 12)

```
CodeBlock (Courier New, Bold 10)
CodeBlock (Courier New, Bold 10)
```

8.1.2. Character Styles

C Code Char (10-Bold, Blue)

Argument (Courier New, Bold Italic 10.5, Magenta-3)

Warning (Calibri italic 12, Red-5)

GUI: Menu, Button (Arial Bold 10, Chart-10 orange)

8.1.3. Italicize

ViewPoint
ViewPoint EyeTracker
ViewPoint EyeTracker Toolbox
Workspace
EyeSpace
EyeCamera
Dynamic Library
dylib
Linux
MS-Windows
Apple Macintosh
Geometry window (the name of the window)