MOUSEY: A multi-purpose eye-tracking and gaze-interaction interface

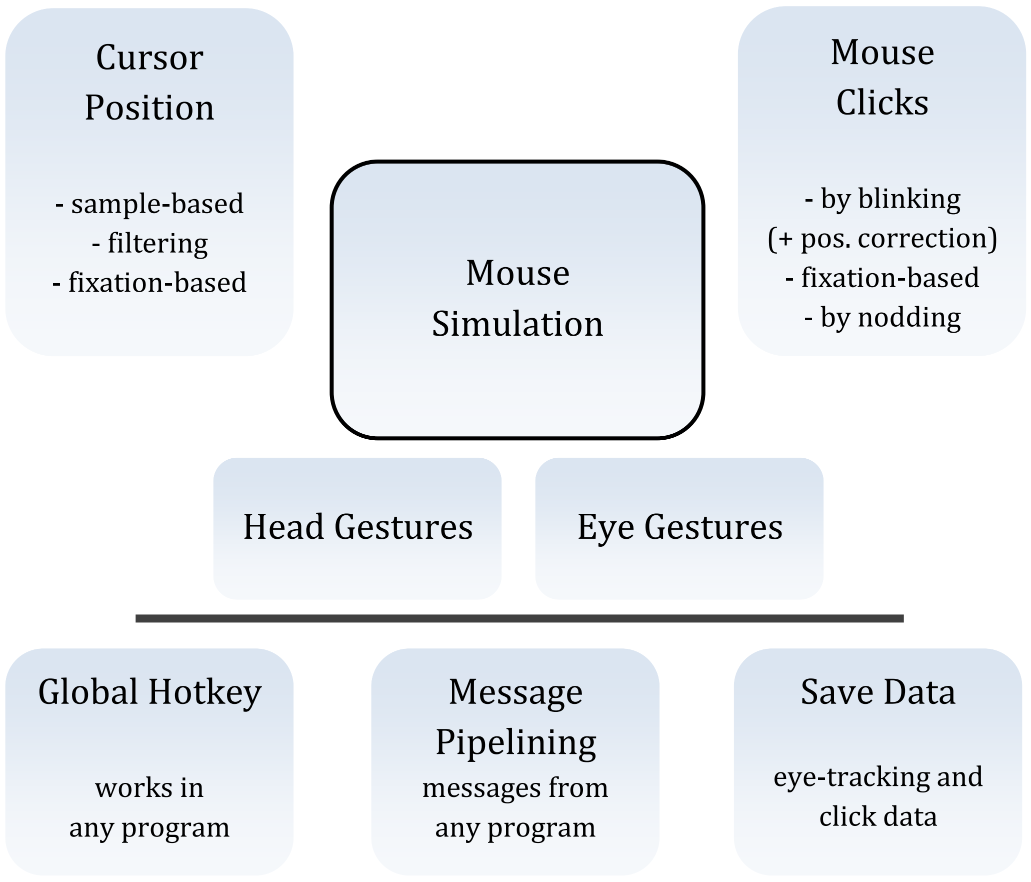
"Mousey is a great tool to easily run experiments and test prototypes of gaze-based interaction – with any software, without limits."

Mousey has been developed by Otto Lutz (email@ottolutz.de), a MSc. student at Technische Universität Berlin.

I would like to thank MSc. Stefan Ruff and Prof. Dr. Matthias Rötting for encouraging me to push on the development and functionality of Mousey – to finally submit it to the student competition of ECEM2013.

Concept / What is Mousey?

Mousey is designed as a general-purpose mouse simulation tool.
It simulates mouse movements and clicks based on eye movements.
So *any* Windows Software which is controllable by mouse input is controllable by your eyes.
The components of Mousey are shown below:



The mouse movement can be simulated by raw gaze position data or via short fixation events.

Mouse clicks can be simulated by fixation events (“look at object to activate”), blinks and nodding of the head. As blinks were reported to feel very intuitive for clicking (see page 6), but cause rapid changes in the gaze vector, a position correction has been implemented.
Additionally to a clicking by nodding, some basic head gestures for keyboard input simulation are recognized.

All mouse simulation can be started and stopped via a global hotkey, which is registered globally with Windows at the start of the interaction. With that hotkey, the experimenter can interrupt the gaze interaction at any time in any windows program.

Additionally, Mousey can receive any textual messages from different applications and send them to the .idf data file. The connection capacities of the supplied RED-oem device are improved, as applications don’t have to establish an UDP connection each.

Installation

If the iViewX SDK (iViewXAPI.dll), RED-m software and .NET framework 4.0 are installed and the RED-m server is set up correctly[1], just start Mousey.exe.
Mousey should work with any SMI eye-tracking system given the data format is the same. It has been tested with the supplied RED-oem and a RED250 system.

Operation

One of the big benefits of the mouse and keyboard simulation provided by Mousey is that designers and developers don't necessarily need an eye-tracker while creating software with gaze-based interaction. It can be tested using normal mouse and keyboard input first, and then swapped to eye control by using Mousey.

Working with Mousey is done in three logical configuration steps: Connection, Mouse Simulation and further options, namely gesture detection, saving data and running different demos.
Please start Mousey to follow these instructions or refer to the screenshot in the appendix.

First, establish a connection to the iView RED-m server.
In the Event parameters you specify the minimum duration and dispersion for the fixation detection of the iViewX. Please note that this affects all position and click simulation. It is advisable to define a rather short minimum fixation duration, e.g. 80-100ms.

For optimal results, calibrate and validate the eye-tracker. The validation result (deviation in X and Y axis) is shown in the status bar at the bottom of Mousey.

The next step is setting up the mouse simulation.

Mouse Simulation

The hotkey is registered globally with Windows, so it works in any program. This feature is important to ensure the investigator/experimenter can activate and deactivate the gaze interaction at any time in every Windows program.
Currently the hotkey is set to the combination WIN + Y[2].

The cursor position on the screen can be set by using raw (sample) data, filtering this data via a Butterworth low-pass or a median filter (default), or by jumping from one fixation to the next.

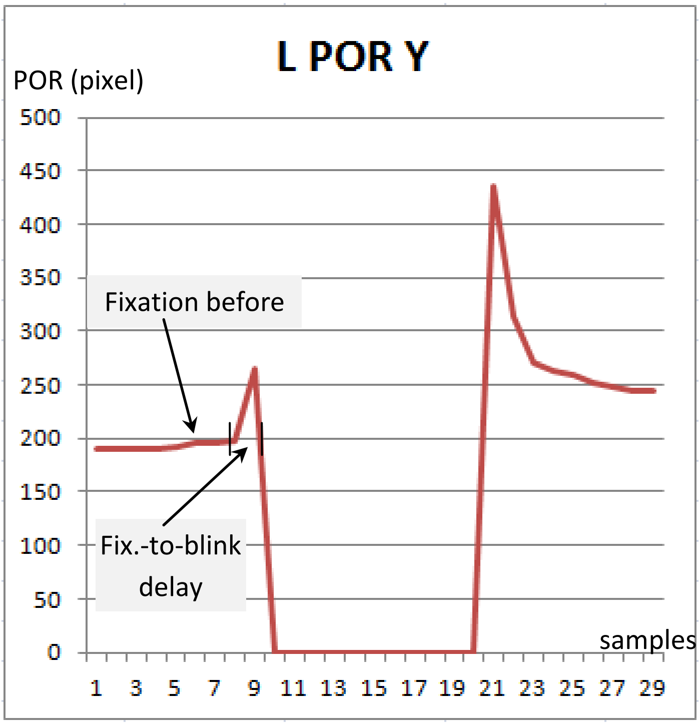
For mouse click emulation, you can combine fixation, standard blink and nodding-based detection methods as you wish. The click via blink with position correction can only be combined with nodding-based detection.

[1] The .NET framework is downloadable at http://www.microsoft.com/en-us/download/details.aspx?id=17718 , the SMI software at http://www.smivision.com/en/gaze-and-eye-tracking-systems/support/software-download.html
[2] Initially it was WIN-X for eXit, but this combination is a protected Windows hotkey in some versions of Windows 7. If necessary, other hotkeys can easily be implemented in the future.

Position correction for clicks by blinking

One of the more advanced features of Mousey is the so-called position correction for blink events. If a person blinks to perform a click, the last tracked gaze vector is further down than the actual element the person looked at (and wanted to activate by clicking).

After evaluating the raw data, it turned out that the curves, which show the distance and speed of the gaze vectors vary quite a lot between subjects. Below you see a typical diagram of the point of regard (point looked at on screen) during a blink.



It is characteristically that the RED-oem eye-tracker shows the gaze vectors going up just before and after the blink, even if the subject stares at a fixed point on screen before the blink.

As all curves vary both inter- and intra-individually, correcting the position of the click solely on the curves needs quite some effort and a bigger data base for statistical analysis.

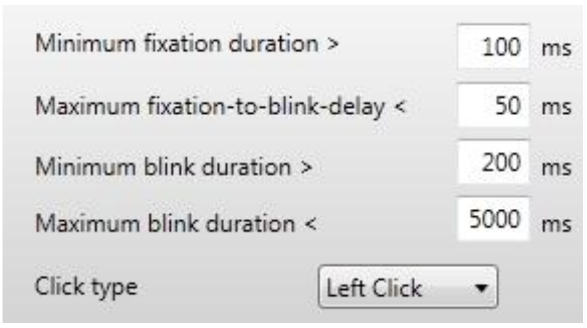
Therefore, the initial ideas of discarding the last n samples before blink or taking an average position of these were discarded.

Our algorithm takes a different approach.

As subjects look at an object before they blink on it (to click), naturally there is a fixation on the object. We take the position of the last fixation before the click as the click location.

When a blink is detected, the mouse cursor is positioned to this location and the click event is performed. For this algorithm, several parameters have to be specified.

It has to be assured that the fixation determining the click position is detected, even if it is short. To make sure no “old” fixation is used, the delay between the end of the fixation and blink must not be too long.



Minimum fixation duration >	100	ms
Maximum fixation-to-blink-delay <	50	ms
Minimum blink duration >	200	ms
Maximum blink duration <	5000	ms
Click type	Left Click	

The default values base mostly on descriptive data analysis of test data sets of 12 people. Further statistical investigation is necessary to determine ideal values for specific user groups.

Gestures, demos and storing data

Eye and head gesture recognition is an interesting research area for gaze-based interaction. With the time limitations given by the ECEM student competition, only basic gesture recognition methods are implemented. In future, these features will be advanced to more powerful methods.

Mousey can detect head gestures to the left, right, top and bottom – and simulate key presses. The timeout determines how long Mousey is blocked from detecting the opposite gesture, e.g. how long the detection of a "left" gesture is blocked after a "right" gesture has been detected. This is so subjects are able to move their head back into neutral position within a given time.

For eye gesture recognition, a slightly modified version of the "$1 unistroke gesture recognizer" by Wobbrock, Wilson and Li (2007) is used.
The eye gesture detection is active as long as the subject keeps looking at the gesture window. This window is augmented with a visual segmentation and reference points for orientation. Though, the strong visual feedback by the points showing the curve of the current gesture seems to distract the user quite substantially – especially when compared to the experimental results of drawing on the screen without visual feedback, outlined on page 6.

For these reasons, the eye gesture recognizer is not linked to any keyboard simulation, but remains an interesting feature to experiment with. The XML files describing different gestures are provided and automatically loaded from the "EyeGesturesXML" subfolder.
Own gestures can be recorded as well – for recognition, they should replace or complement the gesture files provided in the subfolder.

Data recording: Both the .idf file with eye-tracking data and a log-file recording Mousey's clicks can be stored on disk. After clicking "run simulation", you will be asked to provide file names. They will be saved automatically at the end of the simulation, either via clicking the button or by using the global hotkey.

Mousey Messaging pipeline

As the RED-oem only allows one connection at a time, it is not possible to send messages to the data file (*.idf) from other applications. This is especially unfortunate, as new opportunities to realize gaze-based interaction with Mousey need markers in the data during their design phase. To compensate for this, Mousey has an interface to pass through messages (ET_REM) to iViewX and the data file.

Programming details

The interface of Mousey is designed to constantly look for a message being sent to its window handle. This can easily be implemented in any Windows software and every programming language which is able to import methods from a DLL.

This interface is already being used with different Microsoft Office applications, which can be programmed via Visual Basic for Applications (VBA). As VBA has very poor pointer support and Windows 7 has protected memory areas between programs, every character of the message is transmitted individually. In the appendix, there is also sample VBA code which can be used in any MS Office document.

Demo Applications

Two demo applications are supplied with Mousey. The PowerPoint demo includes all necessary explanations. Please assure to activate macros / VBA when you open the document. If opening it via Mousey doesn't work, you find it in the "Demos" subfolder.

The "building blocks" demo is a gaze interaction demo. You can reposition the objects by clicking them, moving, and then clicking again to set them down. Of course, this can be done with any of Mousey's gaze-based positioning and clicking methods.

There is a second mode of operation, which is a test for head gestures. Activate it by pressing "A", and you will be able to move the blocks by head gestures. Selection of the active block is still done by looking and clicking at them, though no "set down" action has to be performed.

With the S, D and F keys, you can swap the mouse cursor or hide it completely. This allows for interesting perceptual experiments, as outlined in the following section.

Experimental Results

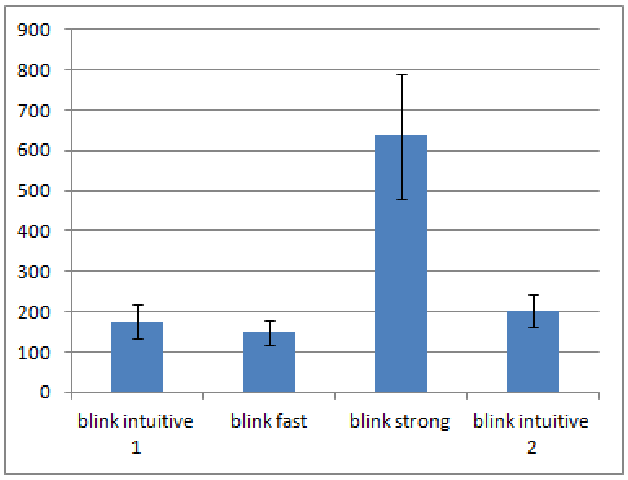
A short study was conducted to gather empirical evidence for suitable default values for the different algorithms and in order to test the acceptance of different gaze-based interaction methods. The sample consisted of 12 participants (m/w) with a median age of 27. During calibration and validation, it was ensured that the deviation (horizontal and vertical accuracy) was <0.5 degrees in both axis.

First, subjects were asked to perform a set of actions as instructed. Each action had to be performed a) intuitively/naturally, b) as fast as possible, c) very emphasized and d) intuitively again. The action set consisted of blinking, nodding, nodding with open eyes (focusing a cross), pushing the head to the left and right ("imagine pushing the cross to the side with your head"), shaking the head and drawing horizontal and vertical lines (up, down, left, right from the center) with the eyes on the screen – without stimulus or visual feedback.

The second part consisted of playing the "building blocks" demo, which tested different ways of clicking - by blinking or by nodding. Another variable was visual feedback – where the mouse cursor was either visible or invisible.

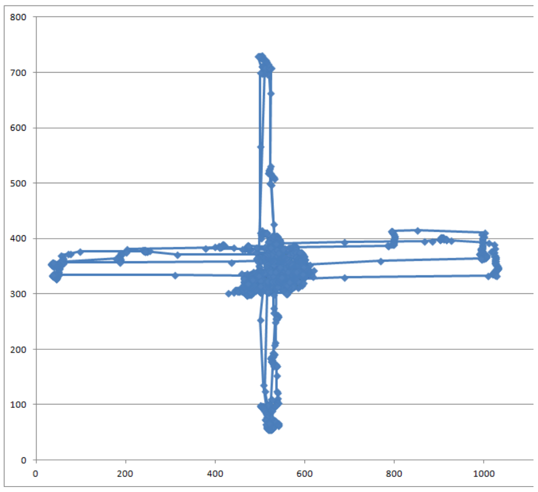
After completion, the subjects were interviewed to give qualitative feedback on how they perceived the different actions and ways of gaze-based interaction.

So far, the data of blinks, “pushing to the side” and drawing lines have been analyzed. Looking at the average blink duration under the different instructions a) to d), it emerged, that the intuitive blink duration is very close to the fastest possible blink (“blink fast”). Bearing in mind that these measures were all about conscious blinks (and unconscious ones are even faster), the minimum blink duration of 200ms for the detection of conscious blinks (versus unintended ones which should not result in any action by Mousey) seems an appropriate parameter. To prevent false positives even better, this value could easily be doubled if people are instructed to blink strongly for the interaction.



Blink duration in ms depending on instruction

For the push gestures, the slopes of different movements were investigated. Maximum movement speeds ranged from about 1 to 6 pixels per sample.
The recorded data, showing typical shapes of the spatial coordinates over time, will help building and testing of gesture recognizers based on machine learning.



Typical result of the line drawing task
[axis: pixel coordinates]

Interestingly, the results of drawing lines beginning from the center in different directions are surprisingly accurate, particularly vertical ones. To the left, there is a plot of an average performer, looking twice in each direction. Bear in mind that there was no stimulus guiding the participants or any visual feedback about the way they looked at the screen.

When trying to reproduce these results with the gesture recognizer (with visual feedback), the results are less accurate. It seems like the visual feedback in the form of points appearing on the gesture detection canvas distracts subjects.

In the qualitative feedback, clicking by blinking and drawing lines on the screen were the most favored ways of interaction. They were described as “intuitive”, “easy”, and “fast”. Nodding while keeping the eyes focused was perceived to be very hard. With the “push to side” gesture, two groups emerged: Some participants loved this gesture for its innovativeness and it “being funny”; others stated they were unsure what to do and didn’t like the amount of head movement involved. Interpreting this, we have to keep in mind though that there was no visual feedback for this action. The results might look totally different after providing visual feedback and implementing a gesture recognizer which can process gestures adaptively, i.e. moving the object a distance relative to the amount of head movement.

There was an interesting result with the building blocks demo: Several of the participants liked the variation without a cursor the most. They stated, the cursor would only distract them.
However, they stated as well that they missed some additional visual feedback concerning activation, hover and closeness to the objects’ centers. These results might be valuable for implementing a new gaze interaction system.

Outlook – next steps

The implementation of more advanced head and eye gesture detection is the logical next step of development. If I am one of the final four participants and get invited to the ECEM, I will advance Mousey’s gesture detection for the conference.

A point to note is that multi-monitor set-ups aren’t fully supported yet. Everything is shown and calculated in relation to the primary screen. After obtaining a second monitor, I will be able to implement full multi-monitor support.

Additionally, I am in the process of implementing a gaze-contingent display simulation, which feels like an overlay over the windows screen: configurable, semi-transparent and with a fully transparent “hole” at the current point of regard. Though with the .NET environment there are some limitations: either one can project an overlay, which changes dependent on mouse movements, or it can be designed to be a click-through canvas. The combination of both is not possible. I will look into possibilities realizing this functionality using DirectX.

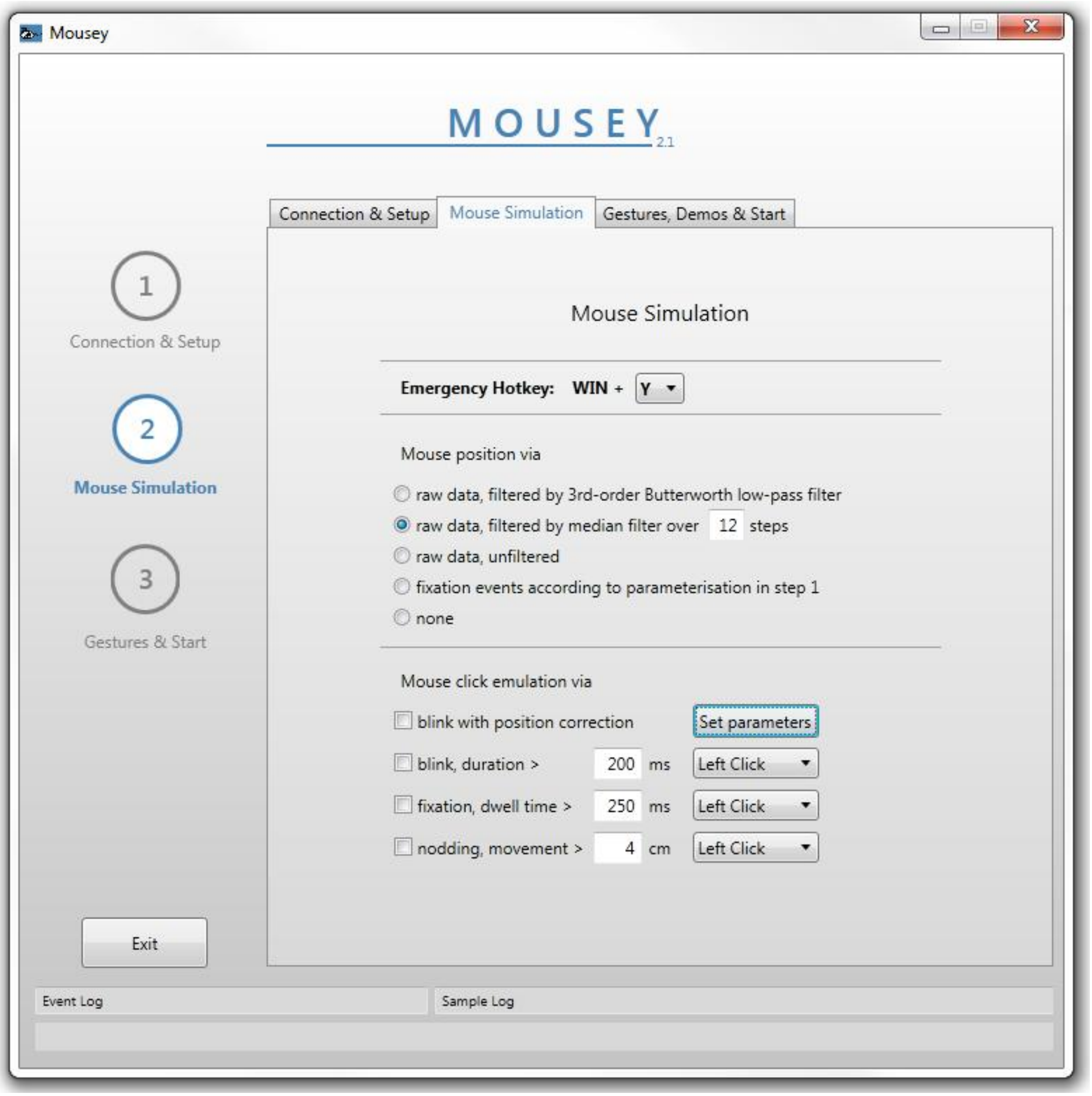
Finally, I am about to use Mousey for what it has been created for – designing, developing and testing new ways of gaze-based interaction.

Appendix

Sample VBA code for sending text to the mousey messaging pipeline

The method `FindMouseyWindow` has to be called once before sending, so the window handler of Mousey is known.

```
Dim Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
  (ByVal hwnd As Long, ByVal wMsg As Long, ByVal wParam As Long, ByVal lParam As Byte) As Long
Dim Declare Function FindWindow Lib "user32" Alias "FindWindowA" _
  (ByVal lpClassName As String, ByVal lpWindowName As String) As Long

Dim Mousey_hWnd As Long

Public Function SendMessageViaMousey(text As String)
  Dim lReturn As Long
  Dim ctr As Long
  Dim lentxt As Long
  Dim byteArray() As Byte

   byteArray = text
   lentxt = Len(text)
   lReturn = SendMessage(Mousey_hWnd, 23231, lentxt, byteArray(0)) ' 23231: start message

   If lReturn <> 0 Then
        MsgBox "Error sending message to Mousey!" + vbNewLine + "Return Code: " + Str(lReturn)
        SendMessageViaMousey = 0
        Exit Function
   End If

   For ctr = 1 To (lentxt - 1)
        lReturn = SendMessage(Mousey_hWnd, 23232, byteArray(ctr * 2), 0) ' for ASCII chars
   Next ctr

        lReturn = SendMessage(Mousey_hWnd, 23233, 0, 0) ' 23233: end message
        SendMessageViaMousey = 2
End Function

Public Sub FindMouseyWindow()
Mousey_hWnd = FindWindow(vbNullString, "Mousey")
   If Mousey_hWnd Then
      'MsgBox Mousey_hWnd
   Else
      MsgBox "Mousey not found!" + vbNewLine + "Mousey has to be running already!"
   End If
End Sub
```


Mousey

MOUSEY 2.1

Connection & Setup | Mouse Simulation | Gestures, Demos & Start

1 Connection & Setup

2 Mouse Simulation

3 Gestures & Start

Mouse Simulation

Emergency Hotkey: **WIN** + Y

Mouse position via

- ○ raw data, filtered by 3rd-order Butterworth low-pass filter
- ● raw data, filtered by median filter over 12 steps
- ○ raw data, unfiltered
- ○ fixation events according to parameterisation in step 1
- ○ none

Mouse click emulation via

- ☐ blink with position correction [Set parameters]
- ☐ blink, duration > 200 ms Left Click
- ☐ fixation, dwell time > 250 ms Left Click
- ☐ nodding, movement > 4 cm Left Click

Exit

Event Log | Sample Log