An Alternative HCI Using Eye gaze and Hand Gestures

Shagen Djanian & Marike Koch van den Broek

Mini-project Report

PMMMSA Autumn 2018

Aalborg University
Vision, Graphics and Interactive Systems

Contents

Chapter 1

Introduction

Interaction that is not based on tactile interfaces and does not require special tokens or wearables enables a larger feeling of flexibility and freedom in the interaction. This mini project strives to create a simple prototype of such an interface for Human-Computer Interaction (HCI). The intended use case is to be enable manipulation of windows in a standard GUI on a Personal Computer solely by the use of the eyes and hands but without any physical contact to any interface.

Chapter 2

Methods

The work of this mini project will strive to implement an intuitive means of interaction and manipulation of windows in a standard graphical user interface of a personal computer. The system will be multi-modal and implemented by utilising tracking of the eye gaze and recognition of hand gestures. The tracking of the eye gaze is done using the Tobii X2-30 eye tracker which is a small, slim tracker which can easily be mounted in a specific place in respect to a screen. The eye tracker has a frame rate of 30 Hz or 60 Hz [Tobii].

For the purpose of detecting hand gestures two methods are investigated. Firstly the use of an open source library called Openpose for processing of RGB images from a webcam is investigated. Secondly, the use of the sensor unit Leap Motion with the related SDK is investigated. The Leap Motion has NIR LEDs and stereo cameras in order to determine position of each pixel in the 3D space above the sensor.

Figure 2.1: Exploded-view of the Leap Motion, exposing the NIR LEDs and the stereo cameras.[Leap Motion, a]

The main framework of the system will be programmed in Python. Furthermore, specific libraries will be used in order to interface to the GUI of the computer opera-

tion system. It was considered to use iMotion for integrating the different modalities, however, since there is only one laptop running the software at university it might be a logistic problem to use the laptop and coordinating with other groups using the system. Also, a group using the software reported technical problems which lead to the conclusion that our time might be better spent looking into alternative frameworks. It was concluded that it would be easier to use Python, which is open source and provides great flexibility but does introduce more requirements for programming skills than iMotion.

Once a satisfactory prototype of a system has been implemented a small user testing will be conducted in order to examine the usability of the methods of interaction.

Chapter 3

Implementation

In this chapter the implementation work carried out throughout this mini project is documented. The implementation is described in three sections; one dedicated to the gesture detection, one dedicated to eye tracking, and finally one dedicated to the framework for combining the parts and interfacing to the GUI.

3.1 Gesture Detection

In this section the work of implementing gesture detection is described. First, the investigation into the software Openpose.

3.1.1 Openpose

The library Openpose is created by the Perceptual Computing Lab at the Carnegie Mellon University in America. It is an open source library available on Github.[Openpose, a] The software is able to detect the joints and links representing the torso and the limps from in an RGB image. Furthermore it can detect the bones of the hands and different parts of the face. It is able to detect in real time if run on a device powerful enough. The installation of the library and the library itself has several dependencies such as Homebrew, Caffe, Cmake, OpenCV etc. The installation is rather cumbersome. The software comes in versions to run on either CPU or GPU. Because the intention is to be able to run a demonstration at the presentation of the mini project it was desirable to implement a system which can run on a laptop. The GPU version is too demanding to be run on a laptop therefore the CPU version was used. The CPU version, however, is still quite demanding. The recommended system specifications are at least around 8 GB free RAM, and a CPU with at least eight cores. Openpose was run on a macbook pro with 8 GB RAM in total and only two cores.[Openpose, b] This laptop obviously does not fulfil the requirements and as a consequence the software was running unsatisfactorily slow. The frame rate was lower that one frame per 2 minutes.

Since it is not possible to make Openpose run reliably on a laptop, a different mean of detecting gestures was investigated. This is described in section 3.1.2.

3.1.2 Leap Motion

In stead of using Openpose the Leap Motion was investigated as an alternative. The Leap Motion is supposed to have a frame rate of 20-200 fps, however, this is dependent on settings as well as available computing power. Nonetheless, this should be better than what Openpose could provide. Also, the requirements for running Leap Motion is much less demanding than for the Openpose. Only 2 GB RAM is required and processors such as Intel i3, which only has two cores.[Leap Motion, c]

Figure 3.1: Example of tracking of the hands using the Leap Motion.

The software used to interface with the Leap Motion was the SDK 3.2 Beta by the company Leap Motion. A newer software, Orion 4.0.0, is available, however, this version does not contain Python interfaces which is needed in order to make an integration in Python. The version of the SDK with the Python interface is only compatible with Python 2.7 and therefore dictates the version of Python to be used.

Using the SDK the output obtained from the Leap Motion is a unique class containing the information of each link in the hand. Each detected link is specified by a name, and vectors describing starting point and a end point as well as a direction of the link. Figure 3.2 shows an overview of the links (bones) the Leap Motion tracks and how they are named. The independent links in the same finger always shares the point in the shared joint. The vectors containing the information can be converted into ordinary NumPy vectors using a class specific function contained in the SDK.

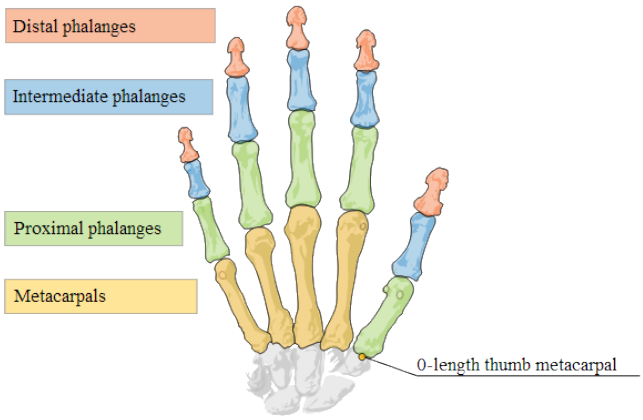

Figure 3.2: An over view of the different links of the hand detected by the Leap Motion and their naming.[Leap Motion, b]

In the SDK part not used for Python there are some detectors included which can detect some higher level configurations. However, when using the Python interface one has to manually create The obtained information vectors are used as the basis for analysing in which pose the hand is. Initially, only one hand pose was made, namely the "Fist". The way the fist is detected by looking at the angle between the metacarpal and the proximal links on each finger. The angle is calculated based on the direction vectors, which are always oriented from the proximal end of the link towards the distal end. This means that the angle between the metacarpal link and the proximal link increases when going from an extended hand to a fist. In practice the average of the angle between the two links across all fingers (except the thumb) has to be above a certain threshold set to be 1.3 radian for the gesture to be recognised as a fist. This threshold was set by making a a fist and various variations of a fist while monitoring the the average angle. This implementation means that no matter which configuration the thumb is in, if the four other fingers satisfy the threshold it will be considered a fist.

Because the detection is a bit unstable and fails to make the right recognition every now and then, a filter was applied. The last 5 samples are stored as a fist or not fist (True or False). The filter is an averaging filter and if the filter output is above 0.5 a fist is recognised. The filter in practice works as a majority vote among the last five detected samples. This method also introduces a small delay when changing from one gesture to another because it will take a few samples before the majority vote outputs the correct decision.

3.2 Eye Tracking

This section describes how eye tracking was implemented.

The device used for eye tracking was as mentioned in chapter 2 the Tobii X2-30. See Figure 3.3.

Figure 3.3: The Tobii X2-30 Eye Tracker.[BU]

In order to interface to the eye tracker the Python SDK from Tobii was initially used. However, this SDK does not support a continuous stream of data from the tracker, which of course is needed when implementing real time control. Due to this issue a range of other libraries were investigated. In the process the open source library PyGaze was found. PyGaze is compatible with Tobii, SensorMotoric Instruments (SMI), and EyeLink eye trackers. The PyGaze library builds on top of the eye tracker specific libraries, in this case the Python SDK from Tobii [**?**]. Using the PyGaze library it is possible to obtain a continuous data steam from the eye tracker.

Once a continuous stream of data was obtained it was clear that the data received was somewhat unrealistic. The assumption made was that it might be because the eye tracker was not calibrated and therefore the device was calibrated using a calibration functionality from the PyGaze library. After calibration the data obtained is much more realistic and shows eg. correlation in the data from gaze on neighbouring corners of the screen. The data, however, does show quite a lot of small variation. This is due to the fact that the eye tracker is sensitive enough to detect most of the small movements of the eye. This is not feasible for stably controlling eg. a cursor on a screen. Therefore an averaging filter is applied averaging across the last 30 samples obtaining a set of mean coordinates which is then used for the control.

3.3 Combined Framework

This section describes the work conducted on interlinking and combining the system as well as interfacing to the GUI of the operating system. The idea for the main structure of the system is as depicted in Figure 3.4. This section is mainly focused on the frameworks for the Python platform (the green box) responsible for generation outputs based on the inputs from the gesture recognition and the eye tracker.

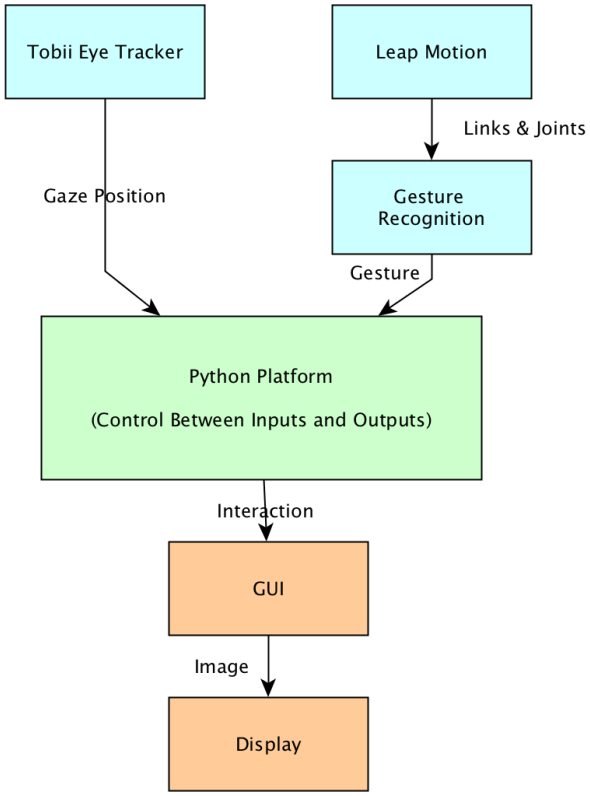

Figure 3.4: An overview on the intended structure of the combined system.

The development work was carried out step-wise each building on top of or improving on the previous step. In the following each step is referred to as an iteration and will be described in its designated section. Initially implementing the manipulation of simply moving a window in the OS GUI is the goal.

3.3.1 Iteration One

This very first iteration is an simple initial attempt to manipulate the windows of the GUI based on a given position and a keyboard input. In later stages this will be the eye gaze position and a specific gesture.

In this simple version the position used is the position of the cursor. In the implemented code a "left-click" is simply performed at the position of the cursor when a specific keyboard button is pressed. If one wants to move a window using this system, one has to gaze at the top bar of the window and push a button in order to "grab" the window. The window will then follow the gaze of the user to a new position in the GUI. This implementation has some inconvenient traits. First of all, the natural way to interact with windows in a GUI using the traditional interfaces is to only gaze at the top bar in the beginning in order to navigate the cursor there and

then "grab" it. After it has been grabbed the user tends to observe the window as a whole and follow it with the eyes to find the right placing or to gaze at either the point they want to move the window to or at points along the edge of the window in order to align it with something. The issue here is that the window top bar at all time will move to the gaze point and thus making it impossible for for the user to e.g. view the edges in order to align. Likewise, the user is not able to observe whether the window as an entity is placed at a desirable spot.

The described system was implemented using the "pyautogui" python library for obtaining the cursor position as well as performing the "click". The library is beneficial to use as it is cross platform. Furthermore the "keyboard" library was used to obtain the pressing of the button simulating the gesture.

Second, the more intuitive way to interact with the window is to not require the user to gaze at the top bar in order to execute the manipulation. This thought leads to the alterations made in Iteration Two described in section 3.3.2.

3.3.2 Iteration Two

In this iteration an alternative interaction scheme was implemented in an attempt to improve on the initial system.

The main idea behind this alteration was that it might be more intuitive to be able to use a specific gesture to "grab" the GUI window no matter where in the window the subject is gazing. Here the gesture was still simulated by the pressing of a keyboard button, but when the button is pressed it is not simply a click. In this implementation the signal of a pressed button initiates a click via the "pyautogui" in order to activate the window currently under the gaze position. Afterwards the window is selected through the use of the "win32gui" library and a function in the library for moving windows is used. The function moves the window to the position of the gaze, which leads to the same inconvenience as describe in **??**. However, in an attempt to overcome this an offset is actively introduced corresponding to the offset of the initial gaze position in relation to the origin of the window. The offset is found subtracting the position of the upper left corner obtained using win32gui from the gaze position. This method works well in relation to obtaining a feasible positioning behaviour corresponding the gaze position, however, the the method also introduces some visual annoyance factor. Because of the way the method selects the windows it continuously selects and releases the windows when a window is being moved continuously according to the eye gaze. This gives a flickering effect of the window which is quite disturbing. The flickering is a consequence of the behaviour of the function used from win32gui, which also contains a function which should disable this behaviour. However, when this function was tested it did not resolve the problem.

As in the previous iteration the libraries "pyautogui" and "keyboard2 are used but as mentioned this iteration also includes the Windows OS specific "win32gui" library. This library is used in order to access the GUI windows and manipulate them through existing functions.

The third iteration described in section 3.3.3 focused on improving the solution by finding a way to eliminate the flickering of the window while moving it.

3.3.3 Iteration Three

In the third iteration the use of the keyboard for input is neglected and instead the a recognised gesture is used. At this stage the gesture detection simply output a boolean indicating whether a fist gesture is detected or not. When a fist is detected a click would be done at the gaze position in order to activate the window at the gaze position. Subsequently the positions of the of the upper left corner and the lower right corner of the window are found using a function from the win32gui library. Using the positions the offset of the gaze according to the upper left corner is found as well as an offset from the upper left corner to a point on the upper bar of the window where you would normally click with the cursor. The latter was initially set as a fixed hard-coded offset, which would generally align with a grab-able point on the upper bar. Later the offset was changed to a more dynamic implementation. Here the offset in the vertical direction was still hard-coded as the height of the upper window bar is usually fixed. However, the horisontal offset was dynamically calculated to be 0.5 of the window width such that it will adapt with the varying sizes of windows.

3.3.4 Iteration Four

In this iteration the eye gaze information obtained using the Tobii eye tracker is included as input. Using the PyGaze library and the methods mentioned in section 3.2 a gaze position is continuously obtained and used as an input for the implemented framework. However, when using the library some issue occurred. The PyGaze library is somehow incompatible with the "pyautogui" library and was giving error messages about some fundamental language issues. After some investigation it was discovered that is was a known issue with a library used by PyGaze. It is claimed to have been fixed but in practice the error still occurs. The issue is resolved by neglecting the "pyautogui" library and instead using the "pynput" library, which essentially works the same. Unfortunately the offset implemented in section 3.3.3 did not work as intended when combined with the eye gaze. The offset was discarded to get the window to follow the gaze. This causes the user to actively look at the top bar of the window to grab it.

Once this iteration is finalised the result is a system which includes both the eye gaze position and the recognised gesture, and which complies with the structure depicted in the diagram in Figure 3.4.

Chapter 4

Tests

This section describes the simple testing conducted in order to evaluate the implemented system as well as the results of the testing.

As mentioned in chapter 2 it was the intention to make a small session of user testing, however, the system in its current state is not at all stable enough to gain any usable information about usability and convenience from test subjects. Therefore a user testing was not conducted on the current system but would be relevant to do on a later stage it the work with the system continued and resulted in a more stable version of the system. In the mean time this section will describe some of the observed issues with the system during pilot testing.

4.1 Eye Tracker

The eye tracker need to be calibrated before each use, which is a cumbersome task to do before actually interacting with the GUI. The users head will also need to be stable during the interaction, which can feel forced and unnatural and be uncomfortable for longer duration of time. Furthermore, if the user is wearing glasses the eye trackers loses precision, which makes the system even more difficult to use. This is a problem as the precision is crucial when simulating mouse movements. This makes it very difficult to click on the intended target causing many miss clicks and annoyances.

4.2 Gesture recognition

The Leap Motion works well enough to recognise simple gestures but the area it detects effectively over is rather small, at least at the natural distance. When focusing on something on the screen for a longer duration of time it was noted that the hand often relaxed causing either the fist to open slightly or drift the hand away from the working space of the Leap Motion causing the user to drop the window.

4.3 Moving a window

It was quite difficult to actually look at the top bar of the screen. It mentally challenging to focus both the eyes at the right location while maintaining the hand above the Leap Motion at the right location. Small errors caused the user to catch the wrong objects resulting the grabbing of wrong windows or moving files in the current window. If the calibration was a little bit off this was very visible since there was an offset between where the user was looking and what the user was actually grabbing. Furthermore, after the window is grabbed the movement is not fluid since eye movements are usually not fluid. The the size of the filter also introduces a small delay. If there was no filer the window would jump around wildly and be uncontrollable but the larger the filter the slower and more fluid the motion was. When trying to just place the window at a location it would oscillate a bit and never actually stop moving. Focusing for the whole movement also proved a bit more straining and unnatural than expected.

Chapter 5

Discussion

This chapter evaluates the implemented system through discussion of methods and algorithms used.

In this section different choices made in respect to the implementation will be discussed. Furthermore, alternative approaches might be suggested.

In the current implementation gestures implemented for recognition were analysed for characteristics that might be distinctive for the individual gesture. This might angles between certain links or distance between points etc. Those characteristics are then used in order to recognise the gesture. The process of recognising or, in the case of more gestures, distinguishing between gestures was manually programmed using a structure of if statements. For distinguishing between several gestures it might be both a difficult process to analyse the gestures and find features that ensure no interconfusion, and it leads to rather large and complex structures of if statements. In these cases it might be beneficial to use supervised machine learning for recognising the gestures. The method could be something as simple as k nearest neighbour, but it could also be something more complex as deep learning. Nonetheless, this method might be better at handling variance in the appearance of gestures and thus also result in a more robust system. As a consequence the filtering applied to the gesture recognition might not be necessary, which also might decrease the delay.

Another area to discuss is the interfacing to the GUI. The interfacing currently carried out in this mini project makes use of operating system specific libraries, which of course in general is not good practice because the system then is limited to run on a specific operating system. Ideally a cross platform implementation should be carried out.

Another point in respect to the implementation of the GUI is that maybe a thorough investigation into the optimal way to implement the interaction and what functionalities are coupled to which gesture would be beneficial for ensuring intuitive interaction.

The current system is rather limited in respect to possibilities for interaction. Before conducting actual user tests a wider set of interactions with the GUI should be implemented e.g. moving, opening and closing windows, moving files and right clicking on files.

During the development it was decided not to use a commercial platform such as iMotion but instead try to implement something in the Python. The benefit in using Python is of course that it is licence free and free to use. However, through this mini project is was experienced how cumbersome it might be to implement something in Python. When using Python one is dependent on available SDKs for the hardware used that can interface with Python. Because a lot of the devices are commercial the companies behind might not be interested in maintaining Python plugins for their devices but would rather have customers use their own, maybe payed, software. In this project such a case dictated which version of Python to use because the SDK was not updated and therefore not compatible with the newest version of Python.

Another issue which was encountered was the problem with different libraries not working together or being incompatible. This can cause a lot of wasted time for debugging or might in some cases simply be a show-stopper.

The final thing that should be addressed is that it is not easy to ensure synchronisation of inputs and/or outputs in Python. For the application in this mini project this was not crucial, however, in some other applications is will of course be very important to have rather precise synchronisation.

Chapter 6

Conclusion

A system for multi-modal interaction with the GUI on a Personal Computer was implemented. The system utilises eye gaze tracking and gesture recognition in order to enable simple manipulation of GUI elements displayed on a screen. The system is at its current state not stable enough to be tested through user testing. The issues that needs resolving before any testing are the precision of the gaze as well as the stability of the gesture recognition. Furthermore, it would be beneficial to have more possible interactions in order to test the system more thoroughly and get a more realistic impression of the system performance.

Bibliography

BU. Available at: <http://sites.bu.edu/crc/research-resources/technologies/>.

Leap Motion, a. Available at: <http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/>.

Leap Motion, b. Available at: <https://developer-archive.leapmotion.com/documentation/python/devguide/Leap_Overview.html>.

Leap Motion, c. Available at: <https://support.leapmotion.com/hc/en-us/articles/223783668-What-are-the-system-requirements->.

Openpose, a. Available at: <https://github.com/CMU-Perceptual-Computing-Lab/openpose>.

Openpose, b. Available at: <https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/installation.md>.

Tobii. Available at: <https://www.tobiipro.com/product-listing/tobii-pro-x2-30/#Specifications>.