2013-03-19

Face Tracking User Interfaces Using Vision-Based Consumer Devices

Norman Villaroman
Brigham Young University - Provo

Face Tracking User Interfaces Using Vision-Based Consumer Devices

Norman H. Villaroman

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

School of Technology

Brigham Young University

March 2013

ABSTRACT

Face Tracking User Interfaces Using Vision-Based Consumer Devices

Norman H. Villaroman
School of Technology, BYU
Master of Science

Some individuals have difficulty using standard hand-manipulated input devices such as a mouse and a keyboard effectively. For such users who at the same time have sufficient control over face and head movement, a robust perceptual or vision-based user interface that can track face movement can significantly help them. Using vision-based consumer devices makes such a user interface readily available and allows its use to be non-intrusive. Designing this type of user interface presents some significant challenges particularly with accuracy and usability. This research investigates such problems and proposes solutions to create a usable and robust face tracking user interface using currently available state-of-the-art technology. In particular, the input control in such an interface is divided into its logical components and studied one by one, namely, user input, capture technology, feature retrieval, feature processing, and pointer behavior. Different options for these components are studied and evaluated to see if they contribute to more efficient use of the interface. The evaluation is done using standard tests created for this purpose. The tests were done by a single user. The results can serve as a precursor to a full-scale usability study, various improvements, and eventual deployment for actual use.

The primary contributions of this research include a logical organization and evaluation of the input process and its different components in face tracking user interfaces, a common library for computer control that can be used by various face tracking engines, an adaptive pointing input style that makes pointing using natural movement easier, and a test suite that can be used to measure performance of various user interfaces for desktop systems.

Keywords: face, detection, tracking, filters, accessibility, assistive technology, depth, perceptual user interface, interface design, HCI, consumer devices, computer input, computer vision, Kinect

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Users with certain disabilities may find it very difficult, if not totally impossible, to use standard input devices such as the ubiquitous mouse and keyboard. The user interface (UI) described in this thesis is one in which users can provide pointer and selection control to a computer just by using natural head and face movements captured by vision-based consumer devices. Because it is enabled by computer vision technology, this kind of user interface is sometimes called a **perceptual user interface (PUI)**. This section aims to define the purpose of this research by discussing the real-world problem that drives this research and the technical problems that this research aims to address.

1.1 Real-World Problem

Current solutions for computer control that are available to users who do not have full control of their hands are not perfect. These solutions employ different modalities such as computer vision, physical manipulation, speech, bio-electric signals, and others that are controlled by other parts of the body that these users can control. These different modalities each have their own advantages and disadvantages. From personal use, interviews with users and assistance providers, some of these solutions are not very robust, require intervention of a helper, require expensive and specialized machines, or are awkward to use, although some have been found to be satisfactory. (Betke et al. 2002, 1-10; Gips and Olivieri 1996; Instruments 2012;

Point 2012; WebAIM 2012) For example, the use of mouth sticks or head wands, though cheap, may require help with putting on the device and the movement can be cumbersome. Using bio-electric signals also requires a helper to put on attachments and specially prepared equipment, although this is usually reserved for people who have difficulty moving even their heads. Some users have reported speech recognition failure with various speech recognition programs. Some staff members at the Utah Center for Assistive Technology are of the opinion that Dragon Naturally Speaking is one of the best solutions for using speech recognition for computer control particularly when the software has been sufficiently trained. While it allows direct selection of form controls on the screen conveniently, pointing operations, when necessary, are unnatural and take longer than spatial gestures (e.g. "go down... down more..."). The same staff members also indicated that the best solution they have for head controlled mouse is the HeadMouse Extreme for its combination of value (995 USD), accuracy, and relative convenience. With it, a reflective dot placed on the head is tracked and the device communicates to the computer as if it is a physical mouse. Selection or "clicks" are done with a separate physical switch. The findings of this research should contribute to a solution that is superior to this or to other comparable technology in usability or robustness.

1.2 Proposed User Interface

For users who have a sufficient degree of control of their heads, some of these challenges can be addressed by a readily available and robust face tracking user interface. This form of user interface uses head pose to control a computer pointer and face gestures, along with several other alternatives, to make selections or special commands. Head movement is natural and happens automatically with normal usage of a computer. It should be noted that this user interface is not new.(Gorodnichy and Roth 2004, 931-942; Hunke and Waibel 1994, 1277-1281 vol.2; Varona et al. 2008, 357-374) The advancement of technology may enable such interfaces to be more usable and practical, which is the object of this research.

I set the following design requirements for this user interface which are based on usability issues from existing solutions and present improvements to such. These requirements guided the selection of the design options evaluated and their implementation. These requirements are similar to those in my previously published work. (Villaroman 2012, 297-298)

- Input should be completely controllable by natural face movements that do not require the movement of the torso.
- Feature detection and tracking should be robust and accurate enough that it doesn't require multiple or prolonged attempts to accomplish simple input operations.
- Aside from the initial hardware and software installation, normal usage (including starting and stopping) should not require the intervention of a helper and should only require minimal effort.
- Operation should be unobtrusive and does not involve having to wear a device or a paraphernalia
- Operation should be robust to some variance in the location of the user's face relative to the screen
- Can be used with a wide variety of existing technology and applications
- If necessary, calibration is only done during setup but not regularly under normal operation.

The target system of control is a computer running Windows 7. This platform was chosen because it has a wide user base and it allows the use of various technology options (e.g. MS Face

Tracking SDK, MS Kinect for Windows SDK) in the implementation of the user interface in addition to other popular cross-platform libraries (e.g. OpenCV, OpenNI, etc.)

1.3 Technical Problem

The primary challenges of face tracking user interfaces - or natural user interfaces in general - are that of control accuracy, reliability, and design. This research aims to provide solutions to these challenges by exploring different design and implementation options and how they affect the accuracy and reliability of this user interface. This primary problem can be further divided into technical sub-problems discussed in the following sections.

1.3.1 Control in Higher Resolution Space

Face movement is minimal when compared to the entire field-of-view of the camera and it can be a challenge when data from such a small region will be interpreted to control a pointer in higher resolution screens. For example, at about 2.5 feet from the sensor, yaw and pitch movements of a face of a certain user captured by a 640x480 sensor and that still allows him to view the computer screen without noticeable strain covers about 100x80 pixels in the field of view. (See Figure 1-1) The challenge then is to determine the best way to use the data in this small region to control a pointer in a screen where a 1920x1080 resolution setting is not uncommon.(Villaroman and Rowe 2012, 57-62)

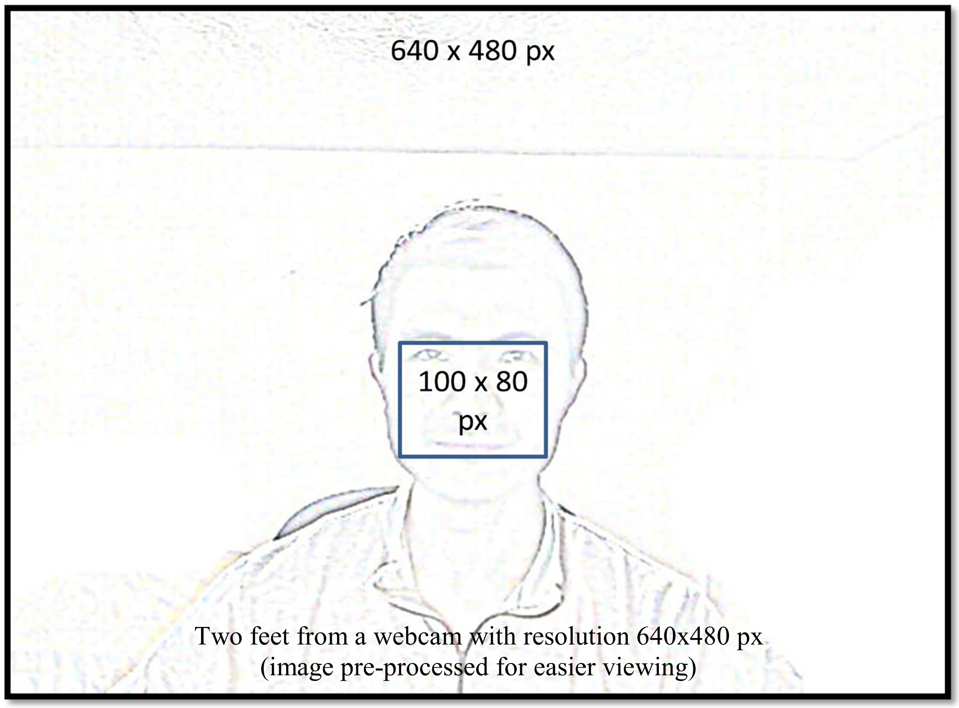

640 x 480 px
100 x 80 px
Two feet from a webcam with resolution 640x480 px
(image pre-processed for easier viewing)

Figure 1-1: Face Area and Movement Span

1.3.2 Noise

Noise is data that is irrelevant to the purpose of a process and causes inaccuracy or poor performance in the objectives of that process. It is naturally introduced in the different components of the user interface. Some of the major sources of noise are discussed in the following sections.

1.3.2.1 Capture Technology

Capture technology is defined here as the set of hardware and software that allows the capture of raw data and processing it for delivery to the next layer in the application architecture. Two classes of technology under this category are regular 2D image cameras for computers (i.e. webcams) and the recently popularized (gwr_press 2011) consumer depth sensors (e.g. *Microsoft*

Kinect, *ASUS Xtion PRO*) that are able to retrieve depth maps of an indoor scene in relatively good detail for their price.

Color/intensity image data is susceptible to illumination variation, camouflaging, and the presence of irrelevant objects. Noise on the pixel location of various objects on the scene is minimal and such cameras can capture images in a wide variety of resolutions.

In November 2010, Microsoft (MS) released a gaming visual sensor for its *Xbox 360* platform called the *Microsoft Kinect for Xbox*, or simply, *Kinect*. This sensor is based on Light Coding technology by PrimeSense Ltd. More about this technology is discussed in Section 3.2.2.2. It produces noise that makes fine movement hard to detect and measure in its generated depth images. This noise is also more pronounced on the edges of objects making edges unreliable for precise calculations. In addition, surfaces that exhibit high specular reflection as well as external sources of IR light (e.g. sunlight) will produce erroneous depth readings.

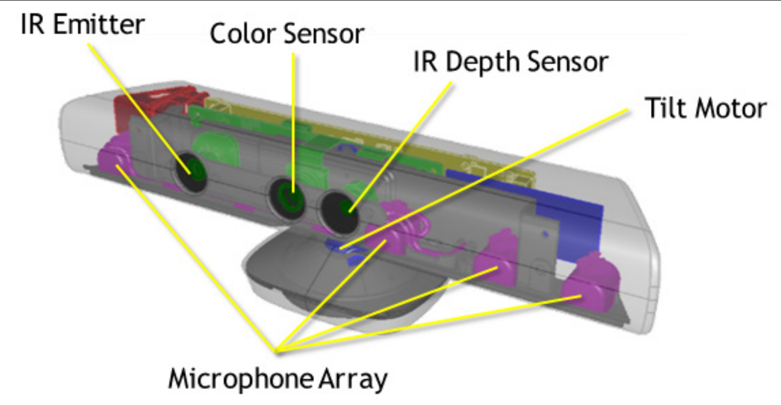


Illustration from a Microsoft web site
(MSDN "Kinect for Windows Sensor Components and Specifications" 2012)

Figure 1-2: Microsoft Kinect Sensor Components Vision Processing

Noise is also introduced by various feature detection and tracking algorithms that are necessary to enable this user interface. Many such methods involve estimation in different ways. Features are represented in ways that are usually different from the raw data (e.g. color histogram features). Various machine learning algorithms involved in many classification methods, by definition, are not able to guarantee accurate classification of data that are not part of the set that they have been trained on. It is also common for such algorithms to have a trade-off between accuracy and speed. So when these methods have to run in real-time as they would need to in a vision-based face tracking user interface, some degree of accuracy may have to be given up.

1.3.2.2 Design and Implementation

Noise can also come from the design of the application and how it is implemented. Some design options might require the use of features that introduce more noise than others. (e.g. Using features in orientation space in addition to or instead of features in pixel space. See Section 3.2.1.1- Pointer Control)

Implementation can also vary in the way noise is dealt with. For example, filtering out noise can be done at different stages in the input process and filtering out later in that process may not be very effective (e.g. filtering the final cursor point instead of the features provided by the face tracking engine). In addition to its effect on speed, the implementation can affect accuracy by the way various data are stored (e.g. data types and structures) and processed (e.g. trigonometric calculation, transformation between different coordinate spaces, etc.). The proper selection and use of other libraries also contribute to the overall speed and performance of this UI.

1.3.3 Retrieval of Feature Characteristics

At the core of a face tracking user interface is its ability to detect face features and accurately track them. Thus, it is essential to identify robust algorithms for this purpose and find or create good implementations of such. Object detection and tracking are widely researched topic in computer vision. While part of the research reviews algorithms and implementations in these categories, it is not intended to use nor personally implement these algorithms to perform a comprehensive comparison. Currently available software libraries that already provide such functionality were identified and included for review in this research. The following is a breakdown of the functionality that may be required in a face tracking user interface.

1.3.3.1 Feature Detection

Face features have to be automatically detected to comply with the design requirements in Section 1.2 - Proposed User Interface. There are different factors involved in the determination of which features to detect. The proper selection of feature/s to detect guides the algorithm implementation and is essential in making it robust. These features have to be represented in different ways (e.g. appearance/image data, statistical data as in histograms, etc.) and the proper feature representation

1.3.3.2 Feature Tracking

While some detection algorithms can be run in real-time such that they would also perform tracking at the same time, the use of a good tracking algorithm may be necessary to make the perceived movement of the face as smooth as the real movement. Tracking algorithms usually use *a priori* information to help obtain a good estimate of the next state of the object.

1.3.3.3 Head Pose Estimation

Some design options may require the ability to estimate head pose or the direction that the head is facing. How this pose vector will be calculated in such a way so as to give accurate and consistent results is not impossible but is still a challenge especially with 2D images where a slightly rotated head (yaw or pitch) can look almost the same as a front-facing head because of the lack of real depth perception.

1.3.3.4 Face Gesture Detection

While pointer control alone can be used to perform selection or special commands by implementing dwell-time selection, it may be more usable to make certain facial gestures be interpreted as commands. In other words, gestures such as raising the eyebrow, blinking, or lowering the jaw can trigger selection or other commands. This problem is seen as an extension to feature detection and tracking so the same algorithms could work here as well. This particular problem is secondary to the other problems listed and is only included for completeness. The effectiveness or usability of various face gestures in triggering various special commands is not part of the research.

1.4 Scope and Limitation

This research is on the investigation of design and implementation options that would make a usable face tracking UI. Various options will be considered. These options do not represent the entire span of possibilities but they represent some of the more common or more sensible candidates.

The focus of this research is on the input side of the user interface. There is no attempt to create an additional graphical user interface (GUI) component or a visual feedback mechanism for it other than what may already be available from the vision processing libraries or the operating system. As an example, a custom on-screen keyboard (OSK) will not be implemented nor a third party version used. As the target system is a personal computer running Windows 7, the built-in OSK will be used for key entry, which is sufficient for the purpose of this research and provides other useful capabilities. (e.g. auto-completion, ability to resize, visibility on top of the Windows logon screen, etc.)

While other modalities such as speech may prove to be a valuable addition to the interface, the only modality investigated in this research is that of perceived face movement. Also, while usability is a major consideration in the direction of this research, for practical research considerations, the focus of the usability tests done is on the ability to point, not on other convenience features, though some measures were taken for those features.

The outcome of this research is a knowledge set of good design and implementation options that is backed by quantitative results. In the process, software prototypes will be implemented that will be used in various tests. Another outcome is the creation and use of a testing process that will allow the comparison not only of the various combinations of options for these prototypes but of other similar UI solutions. To limit the scope, the tests done in this research were done by a single user. This presents some advantages in that the results will less likely be skewed by human factors that can significantly vary from one user to another. However, a usability test that includes several users has additional benefits and is deferred to future work.

1.5 Research Hypotheses

Different hypotheses were formulated regarding the effectiveness of various design and implementation options. These hypotheses are listed below. Some of the terms used are better explained in their corresponding sections in Chapter 3 so they are not discussed here.

Table 1-1: List of Hypotheses

	Hypotheses	**Relevant Section**
1a	State-of-the-art pose estimation makes pointing more accurate	3.2.1.1 Pointer Control
1b	State-of-the-art pose estimation is sufficient for head pose pointing (user input option)	3.2.1.1 Pointer Control
2a	Depth data makes detection and tracking of face features more robust	3.2.2 Capture Technology
2b	Depth data is necessary to make detection and tracking of face features more robust	3.2.2 Capture Technology
3	Tracking is better than rapid detection	3.2.3 Feature Retrieval
4	Current face tracking tech is able to support a robust face tracking UI	3.2.3.4 Face Tracking Engines
5	The Kalman filter deals with noise better than an average filter	3.2.4 Feature Data Processing
6	Pointing operation is easier when noise is minimized	3.2.4 Feature Data Processing
7	A Location/differential pointer is easier to use than a velocity one	3.2.5 Computer Input / Pointer Behavior
8	A differential pointer reduces the data requirements on the face tracking engine	3.2.5 Computer Input / Pointer Behavior
9	External pointing tests can be used to compare performance	3.4 Evaluation
10	Pointing accuracy can be measured by the stationary point spread	3.4.1 Point Spread of Stationary Head

2 LITERATURE AND TECHNOLOGY REVIEW

In this section, both published research and some current state-of-the-art technologies that do not necessarily have associated peer-reviewed publications are discussed.

2.1 Hands-Free User Interfaces

Various solutions exist for users who do not have full control of their hands. Although this research is on face tracking UI, some existing solutions for other modalities are given as they provide different ideas that can ultimately help solve the real-world problem for the user. Note that the literature review for other modalities are not as comprehensive as it is for face tracking since the purpose for including it is that of completeness rather than focus.

2.1.1 Face Tracking

Face tracking user interfaces are not new with some directly related research done on it as early as 1994. (Hunke and Waibel 1994, 1277-1281 vol.2) The application *SINA* (http://dmi.uib.es/~ugiv/sina/) of Varona et al. detected and tracked the nose from image features for cursor control and eye winks from color distributions for control commands (although this wink feature was not observed during testing). (Varona et al. 2008, 357-374) The earlier application *Nouse* (http://nouse.ca) of Gorodnichy et al. tracked the nose and eyes as well and used multiple blinks as control commands. (Gorodnichy and Roth 2004, 931-942) They also proposed and implemented a visual feedback mechanism where a small image follows the cursor and shows how the system is interpreting the current state of the user's head pose. (Gorodnichy 2006) Morris et al. used skin color to detect the face and the nostrils, producing a system that they admit is vulnerable to noise from similarly colored regions. (Morris and Chauhan 2006, 62-80) Chathuranga et al. implemented a system where the nose is tracked for cursor control and where speech recognition is used for selection. (Chathuranga et al. 2010, 359-364) The Camera Mouse (http://www.cameramouse.org/) uses correlation to track user-defined and automatically updating templates to track a small region in a video sequence. (Betke et al. 2002, 1-10) For the same application, Tu et al. used a 3D model fitted to 2D features (using Least Mean Squared Error) to track a face using normalized correlation. (Tu et al. 2007, 35-40)

While this research has the same intended application as some of these mentioned works, it is different in the formal investigation of various design and implementation options of the entire interface. The advancement of technology to its state today provides additional options both for design and implementation as well as opportunities for evaluation.

In addition, some commercial solutions available for this same purpose include the HeadMouse Extreme (Origin Instruments, http://www.orin.com) and the SmartNav (NaturalPoint, http://www.naturalpoint.com). Both track a reflective dot placed on the user's face. HeadMouse Extreme is basically a hardware mouse with a video input, tracker, and logic embedded in the hardware. To the computer it is a mouse so no external software needs to be installed. The SmartNav also tracks a reflective dot but the input simulation is done through the Windows API.

The user interface that is the subject of this study is under this same category. The technologies mentioned here do not completely satisfy the design requirements in Section 1.2

Proposed User Interface. For example, the HeadMouse Extreme and the SmartNav require the user to wear or put on an accessory. CameraMouse requires another person to start tracking for normal operation. The others that were tried were not robust in the way they detect and track face features. Setting aside some usability features, some of the technologies mentioned here are good enough when it comes to performance that they are used for comparison in this research.

2.1.1.1 Eye Tracking

For users with limited head movement, eye tracking may be a better or may even be the only choice. A common challenge for all eye tracking systems is to address the fact that the controller is the same as the visual receptor. Some consequences of this include the partial obstruction of the viewed point by the cursor and the reduced usability or even the total inoperability caused by an unexpected offset of the tracked point. For example, and as personally experienced with two eye tracking systems (Tobii PCEye and Dynavox EyeMax), the pointer , in the process of normal usage, may be set away from the focus of the eye so that one cannot see the object and point to it at the same time, at which point, the pointer would have to be recalibrated.

EagleEyes uses electrodes placed on the face to capture electro-oculographic (EOG) potential and use it to calculate eye gaze.(Gips and Olivieri 1996) Work by Yagi et al. shows more recent analysis in using EOG signals and provides solutions for the drifting and blinking problem inherent in such interfaces.(Yagi 2010, 28-32) Vazquez et al. used infrared illumination where a webcam, infrared LEDs and a 3-axes accelerometer are all put together in a head-mounted system. (Vazquez et al. 2011, 165-170) Reale et al. used images from a two-camera system (one for finding the eye and the other for focusing on the eye) to detect eye gaze.(Reale et al. 2010, 280-285) Heikkil et al. performed a usability study on eye gestures and closures as interaction techniques. (Heikkil et al. 2012, 147-154). Muchun et al. detected eye blinks using pattern matching and optical flow. (Muchun et al. 2008, 351-356)

2.1.2 Other Modalities

Some solutions are not entirely based on vision techniques but employ other modalities. Chathuranga et al implemented a system where the nose is tracked for cursor control and speech recognition is used for selection.(Chathuranga et al. 2010, 359-364) The Vocal Joystick uses acoustic-phonetic parameters which are more appropriate for continuous input to control a cursor.(Bilmes et al. 2005, 995-1002) Inhyuk et al used both image observation and EMG signals from probes attached to the user's neck.(Inhyuk et al. 2003, 1515-1520 vol.1)

Other modalities include physical manipulation using head wands, mouth sticks, manual switches triggered in variety of ways, and others.(WebAIM 2012)

2.2 Capture Technology

Many perceptual user interfaces primarily use image streams from a regular camera, a discussion of which is not necessary here. But recent advances in sensor technology have added a new dimension, literally and figuratively, to vision-based applications that are readily available to consumers. The launch of the *Microsoft Kinect for Xbox* (Kinect) in November 2010 spurred the development of a wide variety of commercial applications and research work. It is based on *Light Coding* technology developed by PrimeSense, Ltd. (http://www.primesense.com/) where structured IR light is used to calculate depth using triangulation. The calculation is done on the hardware side which off-loads depth calculation from the software side. While various depth sensors have been used before, the Kinect made a relatively high resolution (640x480) depth sensor available at a very low price. (PrimeSense 2012) An analysis of the accuracy, resolution, and characteristics of the Kinect sensor itself was made by Khoshelham et al.(Khoshelham and Elberink 2012, 1437-1454) Aligned color images and a microphone array are also usually available in such devices allowing multiple modalities to be used for extended or improved functionality. Since the launch of the original Kinect, a few other similar devices have been made available to the public. (e.g. ASUS Xtion PRO series, MS Kinect for Windows) Various frameworks and libraries have also been developed to process data coming from such sensors. These are discussed in Section 2.4 - Vision Processing Libraries.

2.3 Computer Vision Algorithms

Computer vision algorithms are at the core of perceptual user interfaces. There is a plethora of research done in the different objectives that this particular user interface require - namely, face feature detection, tracking, head pose estimation, and face gesture detection. This review does not intend to give a comprehensive list of research work as that would be practically impossible given their breadth and depth but some are listed that can potentially be useful in face tracking user interfaces.

The classic object detection framework of Viola and Jones which uses cascaded classifiers of Haar-like features using integral images should be mentioned as a landmark work on real-time detection of various objects, including faces.(Viola and Jones 2001, I-511-I-518 vol.1) Modified census transformation has also been used for face detection and tracking.(Froba and Ernst 2004, 91-96; Küblbeck and Ernst 2006, 564-572) Some of these works are the basis for SHORE (Sophisticated High-speed Object Recognition Engine), a proprietary face analysis library published by Fraunhofer IIS. Active Appearance Models have been used for face tracking aided by temporal matching and color-based segmentation.(Mingcai et al. 2010, 701-708) This particular work was used as a basis for the MS Face Tracking (MSFT) SDK.(Smolyanskiy 2012)

In another work, AdaBoost was used for face detection and adaptive particle filtering was used for face tracking.(Zheng and Bhandarkar 2009, 9-27) Local binary patterns and its variants have also been used as feature vectors using various machine learning algorithms for classification.(Di et al. 2011, 765-781; Hongliang et al. 2004, 306-309; Toan Thanh et al. 2009, 1-8) Finally, a combination of some of these methods has very recently been proven to perform face detection accurately and efficiently.(Pan et al. 2013, 12-28)

Some surveys on object detection are also helpful in understanding the capabilities of available methods. Some such surveys follow. Zhang et al. published a survey on face detection in 2010 to update a similar survey by Ming-Hsuan et al. in 2002. (Ming-Hsuan et al. 2002, 34-58; Zhang and Zhang 2010) The survey on object tracking by Yilmaz et al. also contains a general survey on object detection that can help in the selection of which feature to detect. (Yilmaz et al. 2006, 13)

Various tracking methods can be used for faces. Iterative Closest Point (ICP algorithm) has been used.(Smolyanskiy 2012; Weise et al. 2011, 1-10) Particle filters have also been used particularly in one case where tracking hypotheses represent particles and multi-scale elastic matching was used to compute optical flow.(Bhandarkar and Luo 2009, 708-725) Optical flow is also a common tracking algorithm and in one case a pyramidal implementation of the Lucas-Kanade algorithm was used to detect and track faces.(Xiaogang and Yanbo 2009, 89-92)

Some works that attempt to detect and track faces and give an estimate of head pose, aside from those that have already been mentioned include the works of Fanelli et al., Kondori et al. and Weise et al. to accomplish these objectives using depth data. (Fanelli et al. 2011; Kondori et al. 2011, 1-4; Weise et al. 2011, 1-10)

2.4 Vision Processing Libraries

Some libraries and frameworks have been created to help with processing image and depth data for use with user interfaces and other applications. An overview of such libraries is given in this section.

The Kinect sensor was originally designed as gaming paraphernalia for the Microsoft Xbox. A number of months after that hardware release and an outburst in independent research that used it,(Adafruit Industries 2010) MS launched the Kinect for Windows SDK (http://kinectforwindows.com). (Knies 2011) Among many others, it features face tracking, skeleton tracking (presumably based on pose recognition work from single depth images by Microsoft Research (Shotton et al. 2011, 1297-1304)), user segmentation, speech processing using the sensor's microphone array, and various ways to access and register the color, depth, and IR images.

OpenNI (http://openni.org) is an open source framework that provides an interface that connects devices, middle-ware, and applications for natural interaction. It is led by a group of companies as a non-profit organization. It was established after the release of the Kinect which is perhaps a driver for this initiative given that PrimeSense, the developer of the depth sensing technology behind Kinect, is part of this group. A middle-ware library called NITE is provided in this project that features skeleton tracking, user segmentation, hand tracking, and hand gesture recognition.

OpenKinect (http://openkinect.org) and CL NUI (http://codelaboratories.com) are both the products of the leading developers in the initial Kinect hacking contest hosted by Adafruit Industries at its launch.(Adafruit Industries 2010; Villaroman et al. 2011, 227-232) The former is an open-source project while the latter is not although the binary is available. While these provide developers access to Kinect data and various other features, the Kinect for Windows SDK and OpenNI remain as the more stable and more developer-friendly solutions.

OpenCV is an open-source, BSD-licensed library for computer vision and machine learning containing both classic and state-of-the-art algorithms. (http://opencv.org) It has interfaces for various programming languages (C, C++, Python, and Java) and can run in Windows, Linux, Android, and Mac OS. It is popular and widely used in the academe and the industry.

The Point Cloud Library (PCL) is a "standalone, large-scale, open-source project for 2D/3D image and point cloud processing." (http://pointclouds.org) (Rusu and Cousins 2011, 1-4) It started in March 2011 and has continually progressed ever since with regular development contributions from "a large number of engineers and scientists from many different organizations, geographically distributed all around the world". (http://pointclouds.org/about/) It is able to capture and process point clouds from various sensors, including consumer depth sensors as defined in this research.

3 DESIGN, IMPLEMENTATION, AND EVALUATION

The following sections discuss how the entire input stream of the user interface is divided into logical components. Different options are then presented for each component. Some implementation notes are also included on each option to indicate how the prototypes were written. Finally, the evaluation methods for these options are presented which serve as the basis for the quantitative results in the following chapter.

3.1 Development Setup

The prototypes were written in Visual C++ (10.0). The development machine is running on Windows 7 and is powered by an Intel® Core™ i5-2500 (3.3 GHz) quad-core CPU and 8GB of memory. I designed and developed the main input processing component as a shared library (DLL). Different projects representing different face tracking engines are set to use the headers of and link to this main input library. Tests were done on a 23" screen set to 1920x1080. A 2.0 MP Logitech QuickCam Pro 9000 USB camera was used (in 640x480 mode) where a webcam was needed. Either a Kinect for Xbox or a Kinect for Windows (the sensor that is officially distributed and supported for use with personal computers) was used where a depth sensor was needed.

3.2 Input Components

The input stream can be divided into the logical components as in my previous work (see Figure 3-1) but slightly re-worded to be more understandable.(Villaroman and Rowe 2012, 57-62) These are each discussed in the following sections.

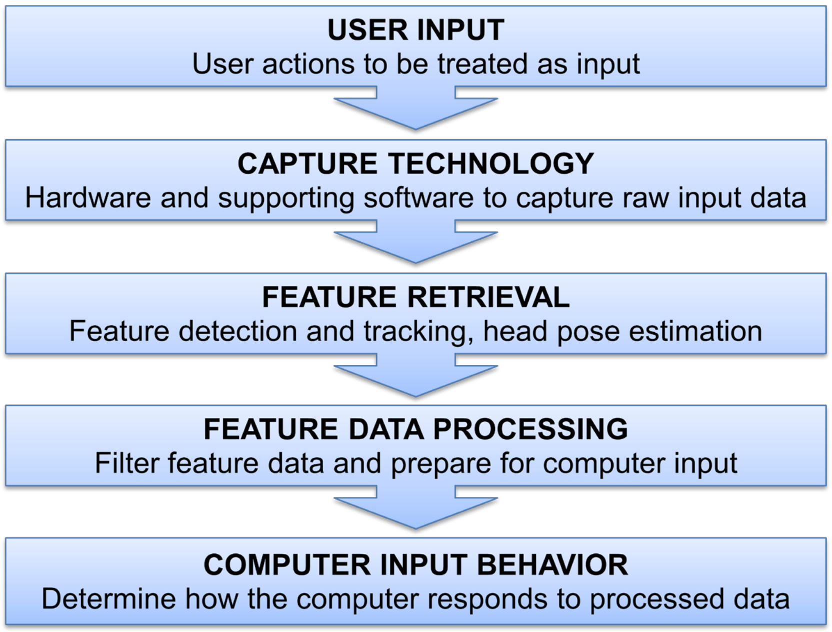

USER INPUT
User actions to be treated as input
CAPTURE TECHNOLOGY
Hardware and supporting software to capture raw input data
FEATURE RETRIEVAL
Feature detection and tracking, head pose estimation
FEATURE DATA PROCESSING
Filter feature data and prepare for computer input
COMPUTER INPUT BEHAVIOR
Determine how the computer responds to processed data

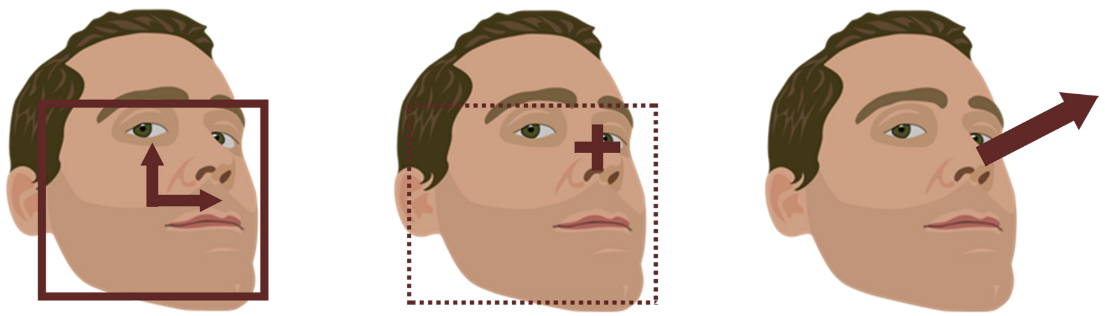
Figure 3-1: Input Components

3.2.1 User Input

This represents how user actions are interpreted as input. The primary control objective for this user interface is pointer control. Selection and other commands are included only for completeness.

3.2.1.1 Pointer Control

For pointer control, there were three different user input options that were considered - point from 2D rotation, point from 2D location, and head pose pointing (illustrated in Figure 3-2 and further discussed below). These options are the primary ones that stand when measured against the design requirements of not having to moving the torso and allowing the user's face to be at any point in the general vicinity of the workstation.



From left to right: 1) Point from 2D rotation (with interaction zone)
2) Point from 2D location (w/o interaction zone) and 3) head pose pointing

Figure 3-2: User Input Options

The *Kinect for Windows Human Interface Guidelines* suggests the use of a physical interaction zone that defines the physical limits of the controlling body part. (Microsoft 2012) This is implemented by imposing limits on the UI when it comes to the generated feature data. This is used when it is possible and applicable. It can be implemented in different ways, depending on the data available and the options selected. The interaction zone appropriately limits the space for relevant data and so it imposes certain accuracy requirements on the user interface. Note that different users can have different interaction zones. In a production system, this interaction zone should be adjustable to meet the needs of different users. For the purposes of testing and development, this research assumes a physical face rotation limit of limit of ~62°

(pitch) and ~39° (yaw) with an aspect ratio of 8:5 calculated from a single user. The virtual interaction zones were set to be smaller than that limit, considering also that the different libraries are not expected to be as accurate as real measurement in their various coordinate spaces.

Hereafter, the term **interaction point** is a pair of values that can be used to directly set the actual pointer location on the screen. Where an interaction zone is explicitly set, the interaction point is defined by the obtained point in reference to the interaction zone.

Point from 2D Rotation - Where head pose is available and accurate enough, the interaction point can be calculated from the yaw and pitch rotation of the face. The interaction zone is defined by imposing limits on this rotation. Since the point is calculated from the orientation of the face, the location of the face is not relevant, thus theoretically allowing the face to be located anywhere in the vicinity of the workstation without affecting pointer control.

Implementation - The interaction zone limits are angle measures representing the orientation of the face with origin at the neutral (straight-facing) position. The default values used for evaluation are ±24° / 0.4 rad for yaw and ±19.2° / 0.32 rad for pitch (MS Face Tracking/faceAPI). Note that for simplicity, the sensor here is at the center of both the aligned world coordinate and the camera coordinate spaces. It is at its neutral orientation (facing the depth axis). In production systems, it is possible for the sensor to have a different orientation and appropriate measures could be implemented to offset this difference.

Point from 2D Location - For face tracking engines that are not able to provide head pose estimates, a tracked 2D point (location) on the image can be used. This location should move changes in the head orientation so this point should be on the surface and not the centroid of the head which is also made available by some face tracking engines. (e.g. MS Face Tracking)

Ideally, another feature data independent from this 2D point (e.g. shoulder or torso) should be available to be a reference for the interaction zone. If this reference point is not available, the interaction zone can just be set initially at the beginning of the tracking sequence and it is fixed for that entire sequence. In the case where the user moves his face outside of this fixed zone, a special command (see Section 3.2.1.2 - Selection Control and Other Commands) may be made to re-initialize the interaction zone. Alternatively, the pointer can be set as a differential pointer (see Section 3.2.5 - Computer Input / Pointer Behavior), which is a setting that simulates having an actual interaction zone.

The 2D location of a tracked point is very common and is used in many face tracking user interfaces (e.g. *CameraMouse*, *HeadMouse Extreme*, *SmartNav*, etc.) and other natural user interfaces in general. As common as this is, using this without any compensation (e.g. setting an interaction zone, setting for how the pointer moves, etc.) can limit usability.

Implementation - Implementing this can be straightforward. All face tracking engines reviewed provide a location for the face. The fixed interaction zone should be set in the same unit as that of the given location. If the point location is given in real-world units, then the size of the interaction zone can just be explicitly set which does not need to be changed afterwards. (e.g. a length of 0.16m in real-world coordinates is theoretically the same at different depths) When the point location is given in camera/view pixels, explicit values cannot be used because it doesn't scale to the user's location so another pixel-based feature could be used for it to scale (e.g. if the size of the head is given by the face tracking engine and it is in pixels, set the interaction zone proportional to the size of the head). If a differential pointer is used, setting this interaction zone is not necessary. Its implementation is discussed in its own section (Section 3.2.5 - Computer Input / Pointer Behavior).

3D Head Pose Pointing - With the advancement of algorithms and technologies that can perform head pose estimation, head pose can be used to accomplish better designs for face tracking user interfaces. (Murphy-Chutorian and Trivedi 2009, 607-626) This option uses the 3D location of the face and the direction of the head pose to calculate where the head is facing or "pointing" to on the screen. This seems to be closer to the ideal and the more natural option. However, this option requires a significant level of accuracy for several reasons. "Head pointing" on the screen limits the total span of rotation because the user can usually rotate his head comfortably to point outside of the screen so there is less input data to process. In addition, depth measurement and head pose estimation need to be more accurate which, though techniques and technology have improved, continue to be a challenge. Head pose estimation will tend to be less accurate than point tracking because the former tracks a larger set of feature vectors.

Implementation - Head pose pointing as implemented here requires the head's yaw and pitch rotation, its 3D location, and the size of the screen in the same coordinate space. Using simple trigonometric rules, the following formula can be obtained:

$$p_x = \frac{z_0 \tan \theta_x + x_0}{screen_x} + offset_x \tag{3-1}$$

In this equation (3-1), p_x is the location of the point that is pointed to along the *x* dimension, x_0, z_0 are the real-world location of the user' head along the *x* and *z* (depth) dimensions, θ_x is the rotation of the head that moves along *x*, $screen_x$ is the size of the screen along *x*, and $offset_x$ shows where the sensor (origin) is located in reference to the screen.

3.2.1.2 Selection Control and Other Commands

Cursor control by itself can provide selection control based on dwell time which is a common option in currently available head and eye tracking user interfaces. Dwell time, of necessity, prolongs the usage of a user interface especially in click-typing on OSKs. It can be problematic with noisy pointer control and/or small targets. This form of trigger is more vulnerable to the "Midas touch" problem, (Jacob 1991, 152-169) where the user accidentally triggers commands from natural movement. Although some research shows that calibrating dwell time on experienced users can increase performance in terms of typing speed, (Majaranta et al. 2009, 357-360) the ability to perform selection or other commands through means other than dwelling on a target can increase usability or at least provides other interaction options. Dwell time can be problematic because it delays the interaction if it's too long or may trigger more unintentional selection if it's too short. Having other options available would cater to the preference of a wider user base that will have a variety of abilities, limitations, and preferences.

Examples of these options include opening the mouth, sticking out the tongue, raising eyebrows, and prolonged or multiple blinks of either or both eyes, among others. However, all face gestures can be problematic because of Midas touch. Some solutions to this include the ability to turn on and off the effect of the gesture as in the Snap Clutch of Istance et al.(Istance et al. 2008, 221-228) and in other commercial eye gaze tracking systems (e.g. Tobii) or use more deliberate gestures as opposed to involuntary or natural gestures. In addition, retrieval and processing of feature characteristics (see Section 3.2.3 - Feature Retrieval) should be sufficiently accurate or these options may render themselves unusable.

Implementation - These options were implemented with the prototype that used MS Face Tracking because the SDK allowed it. It uses a subset of the CANDIDE-3 model, (Jörgen 2001) a parameterized face mask (see Figure 3-3: Candide-3 Face Mask), with six action units (i.e. facial expression markers) and eleven shape units (i.e. face shapes). These action units are *Upper Lip Raiser*, *Jaw Lowerer*, *Lip Stretcher*, *Brow Lowerer*, *Lip Corner Depressor*, and *Outer*

Brow Raiser. For each action unit a numerical value indicates the degree of presence of that marker. In the research prototype, the *Jaw Lowerer* and the *Outer Brow Raiser* were used and were deemed sufficient for the purpose. What has not been fully determined is how the threshold for the value associated with the action units may differ from person to person.

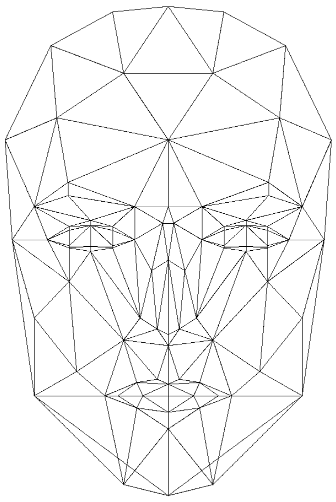


Figure 3-3: Candide-3 Face Mask

Gesture-triggered control can also be accomplished with other libraries if certain facial features (e.g. eyes, eyebrows and lips) are tracked in a sufficiently robust manner. For example, faceAPI is able to do lip and eyebrow tracking but is only available in the commercial version. In any case, this was only implemented as a proof-of-concept so its implementation in the other libraries was not pursued.

3.2.2 Capture Technology

Various devices can provide the raw data for a face tracking user interface. While there are potentially many such, only two classes of vision-based consumer devices are discussed because they are non-intrusive, relatively cheap, and are readily available.

3.2.2.1 2D Image Cameras

A regular camera is used to generate image data used in research and many vision-based applications. In favor of this capture technology is its ability for high resolution capture and its ubiquity, although this is not always taken advantage of because of the resulting increase in computational complexity. Noise shows up in a number of different ways as discussed in Section 1.3.2.1 - Capture Technology.

3.2.2.2 Consumer Depth Sensors

It has been noted that consumer depth sensors such as the Kinect have recently gained popularity and outcomes that have been accomplished or are being worked on are done with the additional depth data available. Depth data generation is performed by parallel calculations of an embedded system ("system on a chip") from triangulation of structured IR light.(PrimeSense 2012) This technology is called Light Coding and was developed by PrimeSense Ltd. Depth images are generated with a maximum resolution of 640x480. A number of different sensors use this technology including the ASUS Xtion series sensors and the MS Kinect (Kinect for Xbox and Kinect for Windows). The cost of these sensors ranges from 150 to 250 USD. In comparison, a PMD Camcube 3.0, a 200x200-pixel 3D time-of-flight camera, was quoted at 6,490 EUR in August 2011. The Kinect sensor has a published range of 0.4-4m (depending on near-mode feature available in Kinect for Windows) (MSDN 2012; PrimeSense 2012) and is designed for capturing indoor scenes. Noise is exhibited as discussed in Section 1.3.2.1 - Capture Technology.

The additional depth data can be very helpful in a face tracking user interface for a number of reasons. Depth measurements are more accurate than any depth estimation method from single monocular images. It also presents an advantage over depth reconstruction from stereo images because algorithms that would otherwise run on the software side for feature identification, correspondence search, alignment, and others are off-loaded to the sensor. It is also not vulnerable to similar textures that make correspondence search difficult in depth reconstruction from stereo images. Segmentation of the foreground also becomes a simpler task by disregarding regions beyond a certain threshold. Because depth calculation uses infrared light, it is able to sense objects in low visible light conditions.

3.2.2.3 Sensor Fusion (Depth and Color)

While there are depth sensors belonging to the same category described in Section 3.2.2.2 - Consumer Depth Sensors (e.g. a certain model of the ASUS Xtion) that only provide depth data, other sensors based on the same technology have RGB cameras that provide color images that can be aligned with the depth data. Using these two data streams together help make up for the weaknesses of the other and provide more information about the scene that can possibly aid in the design and functionality of face tracking user interfaces. In addition, some sensors (e.g. Kinect sensors) also have a microphone array that can perform beam forming (which extrapolates the location of the speech source) that allows them to capture speech in good quality in a possibly noisy environment. The use of sensors with audio capture capability provides convenient hardware extensibility for speech recognition applications.

Various libraries can be used to capture and process data from these sensors. These include OpenNI, the Point Cloud Library, and the Microsoft Kinect for Windows SDK. The MS Face Tracking SDK uses both depth and color streams to track faces.(Smolyanskiy 2012)

3.2.3 Feature Retrieval

As the core of this UI, this component is of utmost importance and requires a high degree of robustness. Anything less robust than this can easily make the UI unusable. Feature characteristics can be collected using various detection and other vision processing algorithms. This is a widely researched topic in computer vision and this research does not attempt to give a comprehensive discussion on it. It is mentioned here for completeness as well as to give a high-level overview of considerations in the selection and use of such algorithms. The following numbered sections will talk about three different computer vision objectives that are needed by this user interface. Surveys on computer vision methods for the corresponding objective are referenced and discussed and some opportunities for novel ideas are mentioned. Various open-source and commercial libraries already have implementations of these objectives and some of them are used in this research. (See Section 3.2.3.4 - Face Tracking Engines)

It should be kept in mind that this user interface operates in a constrained environment where the background generally does not move, the imaging device is stationary, and the user is the primary foreground object. Occlusions other than the possibility of glasses and natural rotational movement will be uncommon and unexpected. Pre-processing methods like background subtraction by image differencing or depth segmentation are definitely helpful. And as a general note, algorithms should also process data with sub-pixel accuracy(e.g. using data types that represent real numbers for point coordinates) whenever possible and appropriate.

Part of the challenge in the implementation is to keep coordinate spaces consistent. Three things define a Euclidean coordinate space commonly used in 3D programming - direction, origin, and units. Different face tracking engines may provide data in different coordinate spaces and the ability to convert them to the needed space while minimizing loss of precision is necessary. As feature data is processed through the line, it may have to be converted into different coordinate spaces. These coordinate spaces should be clearly defined at every point.

The discussion on the different face tracking engines evaluated is in Section 3.2.3.4 - Face Tracking Engines. Because they may be mentioned in the following sections, it should be noted that these options are 1) the MS Face Tracking SDK, 2) faceAPI, 3) discriminative random regression forest (DRRF), and 4) the Haar cascade classifier. In addition, the discussion in the following section is similar to my previously published work. (Villaroman and Rowe 2012, 57-62)

Implementation - Numbers representing feature data use the `float` data type. This type is given an alias that represents different units. (e.g. using **`#define`** to define **`ZeroToOne`**, **`Meters`**, **`ScreenPixels`**, **`Degrees`**, etc.) Custom 2D and 3D structures (e.g. **`Point3D`**, **`PointInBox2D`**, etc.) are implemented as templates so that the type aliases can be used for readability (though they define the same type).

The different face tracking engines use the same input processor implemented as a shared library. Because of this the input processor has to process data in the same coordinate space conventions. The feature data coming from the face tracking engine is converted to a common direction (left-hand rule). Once filtered, it is converted to numbers bounded by 0 and 1 where 0 represents the origin (at top left for the x and y dimensions), which is then further processed.

3.2.3.1 Detection

It is critical to determine which features to use as the primary controller of the user interface. This depends on the quality and kind of data provided by the capture technology used (see Section 3.2.2 - Capture Technology) and, obviously, the performance of the particular detection algorithm. The nose has been recommended as a good feature to track from regular images because of its visibility and consistent intensity profile in various poses.(Gorodnichy 2002, 181-186) Regular images also provide characteristically high intensity gradients for the eyes. However, in a noisy depth image, the eye on its own is harder to differentiate from another relatively flat patch. In a depth image the nose is more prominent.

The feature or a combination of such features to be detected has to be unique in the image and removal of non-relevant data (e.g. by segmentation) would be helpful in making this happen. Using a combination of features can provide confidence levels that can be adjusted to reduce false positives.(Villaroman and Rowe 2012, 57-62)

Using depth data can help simplify the detection problem by providing additional recognizable data. For example, depth data can confirm whether the detected feature is on a head or not, reducing false positives. The use of a combination of detection methods or data streams can help make detection more robust.

The identification of good methods for face detection can be aided by reviewing some of the latest surveys on it, which include works done by Zhang et al. (Zhang and Zhang 2010), Yilmaz et al. (Yilmaz et al. 2006, 13), and Ming-Hsuan et al. (Ming-Hsuan et al. 2002, 34-58)

While the ability to detect per-frame shows the computational efficiency of an algorithm, this typically introduces noise and ignores valuable *a priori* knowledge. Coupled with tracking, detection does not have to be extremely fast particularly when performance is on the line. It is very important that performance is maximized to minimize false positives and negatives. It is a major failure if the system cannot detect the face when it is present. As a comparison, it is one thing if a standard mouse does not work linearly as expected and quite another thing if the user cannot hold the mouse at all.

3.2.3.2 Tracking

If the detection happens fast enough, detection can be used to determine the state of the feature at every cycle. Alternatively, tracking methods can be used that take advantage of *a priori* and current information to get a better estimate of the real state of the feature. Tracking stops unnecessary detection from being performed when the feature has already been detected and it allows the detection algorithm to have space to focus on performance over speed.

Face movement is generally smooth and tracking can help model this more accurately. Yilmaz et al. did a survey on tracking methods in 2006.(Yilmaz et al. 2006, 13) It divided object tracking into point, kernel and silhouette tracking. The survey indicated that point tracking methods find corresponding points in subsequent frames and would usually be appropriate for tracking smaller objects. This may be appropriate if small features are tracked such as the eyes or nose. Kernel tracking methods use the object's shape and appearance and a motion vector is produced from parametric transformation of the object. This may be more appropriate for tracking the head or face. Silhouette tracking can be useful for certain face gesture detection. This survey referred to works that used regular 2D images. Some of them may be applicable to 3D point clouds. However, if depth images from consumer depth sensors are used, the noise in the object boundaries will cause inaccuracy in the results so the use of such boundaries should be avoided.

Weise et al. used the entire face excluding the jaw area for rigid tracking from depth images using Iterative Closest Point (ICP).(Weise et al. 2011, 1-10) They have published a program (currently in beta) called *faceshift* that allows users to model their faces with various expressions in 3D. This model can then be used so that their face movement can animate an avatar with realistic expressions. Use of this program shows the robustness of, among other things, ICP in fine tracking of faces. Interestingly, the same method is used in the MS Face Tracking SDK.(Smolyanskiy 2012)

3.2.3.3 Head Pose Estimation

This section is based on my previously published work. (Villaroman and Rowe 2012, 57-62) If face pose estimation is accurate enough, it can prove to be very useful in processing input. Murphy-Chutorian and Trivedi presented an excellent comprehensive survey of such methods with an annotated comparison of accuracy in 2009 with a separate evaluation for fine and coarse estimation. (Murphy-Chutorian and Trivedi 2009, 607-626) Some of the fine estimation methods in these works show promise. (Balasubramanian et al. 2007, 1-7; Wang and Sung 2007, 1864-1874; Yun and Huang 2006, 6 pp.-8) However, as they noted, many of the works reviewed make assumptions or use methods that make them less applicable in real-world and real-time applications. These include limitation to a single rotational degree-of-freedom, the requirement of manual intervention, the use of test data that is very similar to the training data, and the requirement of specialized setups or non-consumer sensors, among others. They have identified these assumptions and associated them whenever applicable with the reviewed methods. While these methods had been or could be improved by more recent technology, techniques, or datasets, the use of this survey is recommended in the selection of face pose estimation methods particularly because of the comparison on accuracy.

Recent works on face pose estimation using depth information address some of these limitations and, perhaps with some improvement or modification, are appropriate in real-world face tracking user interfaces. (Fanelli et al. 2012; Kondori et al. 2011, 1-4; Weise et al. 2011, 1-10)

Tracking methods may also be applicable and useful here to capture smooth movement and some pose estimation methods already use them.(Murphy-Chutorian and Trivedi 2009, 607-626) A method worth noting for its potential to track face location and pose accurately using 3D point clouds is Iterative Closest Point (ICP) as used by Weise et al. (Weise et al. 2011, 1-10) The performance of their methods can be seen in the use of software based on their research called *faceshift*.

Pose estimation will usually result in or can be derived to an Euler angle (the angular orientation of a rigid body relative to a fixed 3D frame of reference). Yaw and pitch rotations are the primary movements that will control the pointer. The roll rotation is not naturally a pointing movement but may be used for special commands. While the face tracking engines evaluated are not very explicit about how they perform pose estimation if they do, it is known that rotation can be calculated when a model or face mask is known and it can be corresponded with the image using POSIT (pose from orthography and scaling with iterations).(Dementhon and Davis 1995, 123-141; Prasad and Aravind 2010, 162-169)

3.2.3.4 Face Tracking Engines

Instead of implementing some of the techniques mentioned or an improvement of them, which would extend work beyond the scope of this research, it was decided instead to use current and state-of-the-art implementations of good or novel techniques. These different face tracking libraries represent the different options evaluated for feature retrieval and are now discussed. This discussion has been separated from the first three feature retrieval objectives to provide a single discussion for each one because each is able to perform multiple objectives. The comparison among the different face tracking engines are done with the same set of options which include tracking a 2D location instead of the other options that require a head pose, a moving average filter, and a differential pointer.

Microsoft Face Tracking SDK - The Microsoft (MS) Face Tracking SDK is a closed source but publicly available library that uses the MS Kinect for Windows SDK that captures and processes data from an approved Kinect sensor. There are currently two such sensors - the Kinect for Xbox and the Kinect for Windows, the former being the original game console device, and the latter designed for use with a PC with additional capabilities such as near-mode vision. Note that the prototypes made for this research used version 1.6 of the SDK.

Direct hints of the techniques used was given by one of the developers behind it.(Smolyanskiy 2012) It uses active appearance models (Mingcai et al. 2010, 701-708) as the core of its 2D tracker but they have extended it to use the available depth data for more robust detection and tracking. It uses ICP as well for the likely purpose of tracking head movement. The SDK provides the 3D location and the 3D orientation of the head. It also provides the 100 2D features points and the 121 3D feature points of the tracked head. In addition, it also exposes a CANDIDE-3 model (see Section 3.2.1.2 - Selection Control and Other Commands) corresponding to the state of the tracked face.

In addition to the SDK, MS also provides a program called Kinect Studio which is able to connect to the sensor data stream in an application that is using the Kinect for Windows SDK. Among other things, this provides visualization of the various data streams and it provides recording and playback functions of such streams. In this way, raw data can be standardized for various purposes. For this research, it is used to provide the same raw data for evaluating and comparing certain option. (see Section 4.3 - Filters) In addition, it can be used to standardize various behaviors such as making sure that interaction zones defined by orientation limits is the same as interaction zones defined by location limits.

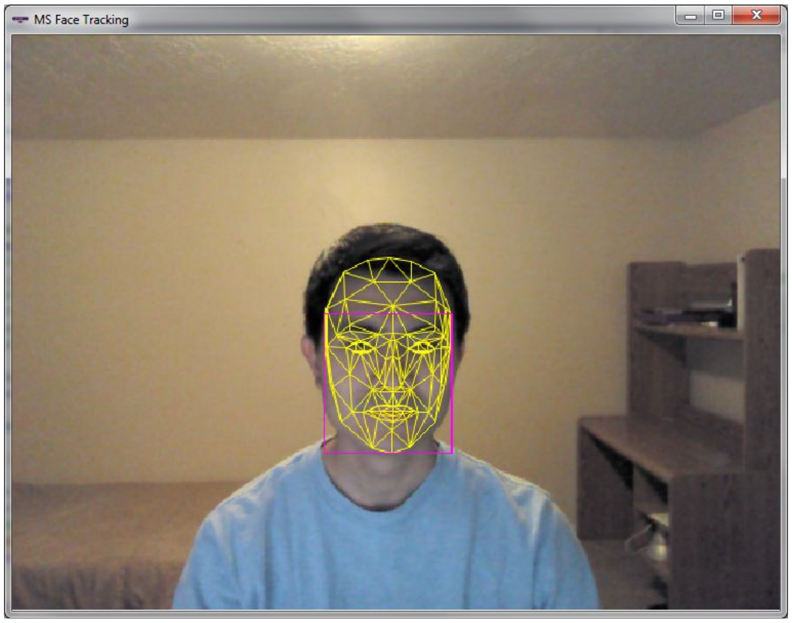


Figure 3-4: Face Detection and Tracking Using the MS Face Tracking SDK

Implementation - The detection happens automatically when **`StartTracking`** is called on a class implementing the **`IFTFaceTracker`** COM interface. This class is created using the factory method **`FTCreateFaceTracker`**.

If the tracking state is successful in the previous frame, calling **`ContinueTracking`** tracks the current state of the face without performing detection on the entire data frame. Otherwise, **`StartTracking`** should be called again and detection is performed. Detection and tracking are implemented to run on a separate thread.

Using the Kinect for Windows sensor (as opposed to the Kinect for Xbox) allows the SDK to detect and track faces that are closer to the sensor than usual (~0.4 - 0.8 m) with its near-mode feature. The SDK has a skeleton tracking ability for upper bodies that helps with face tracking by giving it hints on where to start looking for the head. If no hints are given (it is a parameter passed in **`ContinueTracking`** and **`StartTracking`** of **`IFTFaceTrackier`**), tracking can fail in near-mode and it can be problematic. For this reason, skeleton tracking should be turned on particularly when using near-mode.

The **`IFTResult`** COM interface contains the result of face tracking. The 100 tracked points are obtained using **`IFTResult.Get2DShapePoints`**. These points are in pixel coordinates. Obtaining relative distances in 2D space using pixels coordinates can be problematic. For example, it is difficult to define an interaction zone that scales with the face when the results obtained are in pixel units because the width of that zone may be 150 pixels in one real world location and 100 pixels in another that is farther from the sensor. A work around can be accomplished by using face feature distances (e.g. express the zone as a factor of the distance between the eyes). Alternatively, the tracked points can be obtained with its 3D representation in real-world units by using **`IFTModel.Get3DShape.`** This allows the SDK to determine fixed-size real world regions that are sufficiently invariant to the user location (e.g. a 0.2m-wide 3D region will be still be ~0.2m wide whether the region is placed closer or farther from the sensor). However obtaining this is not very straightforward, at least in the C++ API. The Euler angle representing the head orientation can be obtained from a face tracking result using the **`IFTResult.Get3DPose`** method. It gives this measure in degrees.

SeeingMachines faceAPI - faceAPI is a closed-source and commercially licensed library for face tracking using regular PC cameras. The non-commercial version is able to provide the location and the orientation of a head while the commercial version is able to provide more data such as lip and eyebrow tracking. While no evidence can be found that hints what techniques it employs, it was obviously one of the best choices for a face tracking engine even at first try.

Detection is quick, tracking is smooth and is robust to scale, rotation, and partial occlusion. Because various WDM (Windows Driver Model) cameras can be used with it, it takes into account camera calibration data for more accurate tracking.

Part of the purpose of the evaluation of this library as a feature retrieval option is to support the argument that raw depth data is not necessary for robust tracking. It is able to provide good estimates of 3D head orientation and location from 2D video.

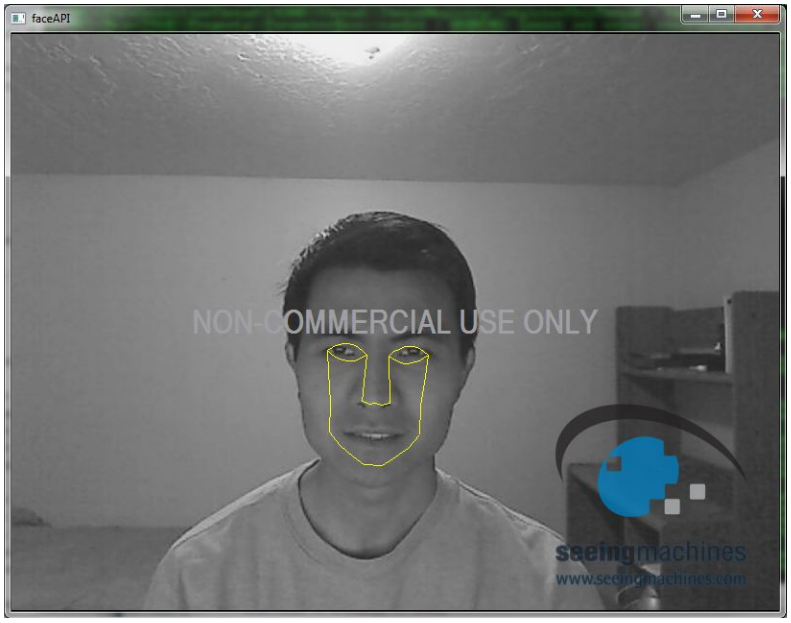


Figure 3-5: Face Detection and Tracking Using FaceAPI

Implementation - Detection starts at the beginning of a tracking sequence which is initiated when **`smEngineStart`** is called. Worker threads are created internally in the engine and call back functions are implemented and bound to those threads. These functions receive the current feature vector. The feature vector includes the location of the head in world coordinates (i.e. sensor is origin, units in meters) and the orientation in radians and is stored in a **`smEngineHeadPoseData`** structure. The SDK provides very convenient conversion methods between points in the world, face (local object), and projective view coordinate spaces. One example of the use of these conversion methods is the definition of the interaction zone by distance limits. The limits can be defined in the face coordinate space (e.g. ±0.06m because the origin is the center of the face). These points can then be converted to the world coordinate space which can then be converted to 2D view coordinates if necessary.

Random Forest - This is the term used in this research for the work done by Fanelli et al on using discriminative random regression forests (DRRF) for head detection and head pose estimation from depth images.(Fanelli et al. 2011) It uses training data generated using a Kinect sensor. Training was done on users from 1m away so that makes classification and regression less robust at a closer or at a farther distance. The data set used had 24 sequences of 20 people. It was annotated using an ICP-based tracker.(Besl and McKay 1992, 239-256) Detection is done at every computation cycle so no independent tracking mechanism was implemented. The best published results of this method indicate a yaw error of 8.9±13.0°.(Fanelli et al. 2012; Fanelli et al. 2011)

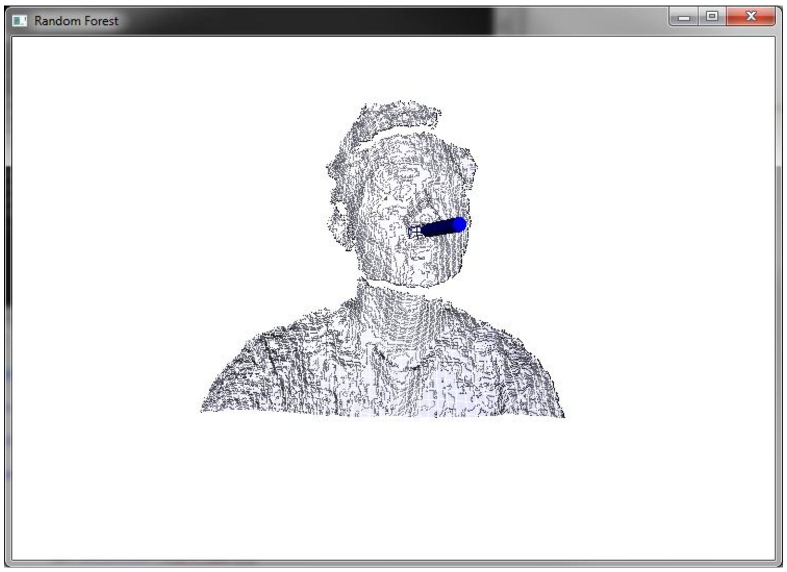


Figure 3-6: Head Detection and Pose Estimation Using Random Forests (DRRF)

Implementation - This algorithm returns the location of the head in real-world units (mm) and the orientation in radians. Since the location of the head is closer to the surface (though not necessarily on the surface and as opposed to being in the center where rotation does not change it), it can be directly used as the tracked point for location based tracking. The stride value, a trade-off setting for performance and speed, can be changed. It defaults to 5 but was set to 10 (faster but a little less accurate) for the tests used here so that it has a frame rate that is closer to the others. The program reads and loads the decision trees included with the program at the beginning. In addition, this can detect multiple heads. For simplicity, the first head is tracked for input.

Haar Cascade Classifier - The object detection framework of Viola and Jones (cascade classification of Haar-like features using integral images) (Viola and Jones 2001, I-511-I-518 vol.1) was a landmark work that made face detection more practical in real-time applications for its speed and effectiveness.(Zhang and Zhang 2010) Because of its popularity and availability, it is evaluated in this study as it provides a baseline method for detection (though a number of other common methods are also available). The implementation in OpenCV was used. Its use of integral images makes feature calculation computationally efficient. Adaboost (adaptive boosting), which combines weak classifiers to produce a more accurate one, was used for training. Classification is done by going through nodes representing the weak classifiers successively. Since detection happens at every computation cycle, there is also no independent tracking mechanism for this option.

It is possible to add to this option the ability to estimate head pose. One such method that is available in OpenCV is POSIT. This was not implemented at this time because only a baseline method for detection and tracking is needed.

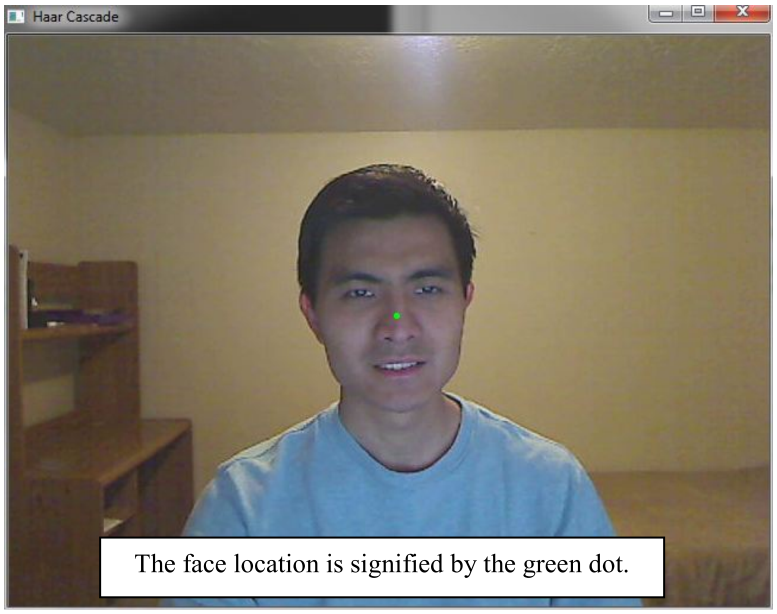


Figure 3-7: Face Detection Using the Haar Cascade Classifier

Implementation - OpenCV has various pre-trained Haar classifiers in XML format that represent corresponding training data for a particular feature (e.g. face, eyes, nose, etc.). The one used for the research prototype is the dataset for faces contained in **`haarcascade_frontalface_alt.xml`**. The implementation is straightforward. A **`CascadeClassifier`** object is first loaded with the data in the said XML file using its `load` method. The incoming video frame is pre-processed by converting it to gray scale and performing histogram equalization on it. The **`CascadeClassifier.detectMultiScale`** is then used to detect faces in the frame. The result is a vector of rectangles that would indicate the location and the size of possibly multiple faces. The size of the expected face was set to avoid false positives on smaller regions which have been observed. This can also detect multiple faces. For simplicity, only the first detected face is tracked for input.

3.2.4 Feature Data Processing

Feature data retrieved from the face tracking engine may come in various forms and conventions. Namely, their points and angles may go with their own coordinate spaces, origins, and units. This data has to be transformed into another form that could then be used directly to send input commands through the Windows API. This transformation includes filtering, scaling, conversion into 2D coordinates, and converting to screen points. This section will primarily discuss filtering as the other methods are straightforward enough that a discussion is unwarranted. The conversion to screen points is discussed in Section 3.2.5 - Computer Input / Pointer Behavior.

The collected feature data is expected to have some degree of noise. Without additional processing, this may lead to the control of a pointer that jumps around and on and off the target causing it to be less usable. A number of filters can be used to mitigate this noise. This is more effectively and simply achieved if these filters are performed on the feature data points instead of on the resulting pointer as previously suggested.(Villaroman and Rowe 2012, 57-62) This is so because some combinations of options might use a combination of feature data points which would merge the noise of the individual data points. In addition, the random noise that needs to be filtered is primarily caused by every component discussed up to this point (Sections 3.2.1 - 3.2.3). The noise caused by scaling and other processing as discussed here is not random and is not very significant.

A number of different methods can be used to do this. Among them are mean/median filters, mean-shift,(Varona et al. 2008, 357-374) particle filters, and the Kalman filter. In this study, as it is not a comprehensive study of every option, only two good candidates are evaluated and implemented - the average filter and the Kalman filter.

Before these options are discussed, note that in general, broad pointer movements do not need to be filtered. It is when an object is pointed to that accuracy is more important. For this reason, the filtering is only applied when the face is steady. This is different from previous work where the filter was applied the entire time.(Villaroman and Rowe 2012, 57-62) One reason that this option was pursued is that broad movements can affect the result of the filter adversely (e.g. latency, boomerang effect for predictive filters, etc.)

Average Filter - Average filters are easy to implement and are relatively computationally efficient. However, it takes into consideration the value of all data points regardless of outliers caused by noise. Where there is a series of data provided by the face tracking engine this filter is implemented as a moving average filter. A variation of this can be done by making it weighted, with the most recent points having more influence on the estimated point.

Implementation - The different kinds of points in the prototypes (e.g. **`Point2D`**, **`Point3D`**) implement the member-less **`Filterable`** interface. The **`MeanFilter`** implements the **`IPointFilter`** interface. This filter contains instances of the **`SimpleMovingAverage`** class which stores the values in the sample and calculates the moving average.

The method for checking if the face is steady or not is the same here as in the implementation of the Kalman filter. The current point and the point from ten cycles back are compared, taking into account their limits (because different options will have different values for limits). If the difference is greater than a certain threshold, the pointer is considered moving and the filters are then disabled, cleared of past data, and set to a new beginning value. If the pointer becomes steady, the filters are activated again.

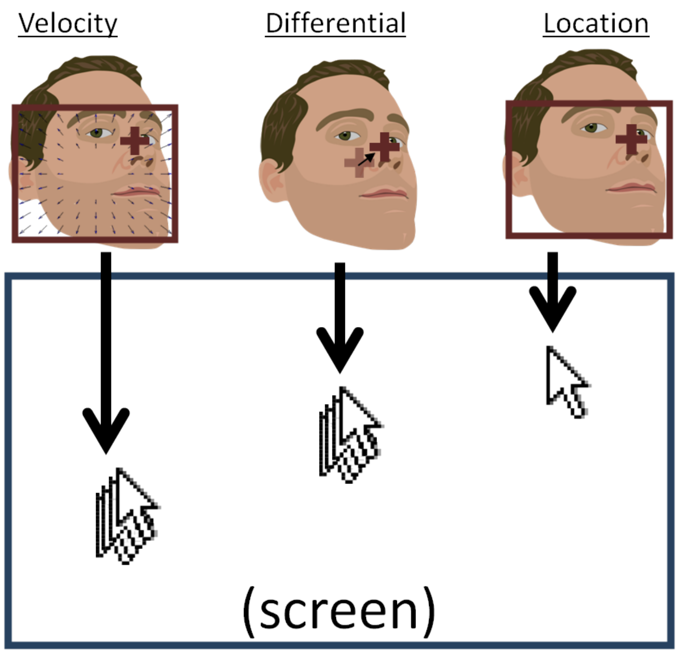
Kalman Filter - The Kalman filter is an appropriate method to use in this scenario, contrary to a previous finding that "they do not achieve good results with erratic movements such as head motion". (Varona et al. 2008, 357-374) Upon further analysis, that conclusion seems to have come from misinterpreted use of the Kalman filter in an evaluation of tracking methods for HCI where the Kalman filter was used to aid in tracking and not to make the pointer movement smooth. (Fagiani et al. 2002, 121-126) The classic Kalman filter can operate on a simple time-discrete linear model such as this one and is relatively resilient to outliers and Gaussian noise in general because such noise is considered in the algorithm. As a recursive Bayesian method, it provides an efficient way of estimating the true state of an object by recursively predicting the next state and updating it with new observations if there are any. It asks for parameters for the process model, control input (which is not necessarily applicable in this case), new measurements or observations, and noise in the process and in the observations. (Villaroman and Rowe 2012, 57-62)

Implementation - OpenCV has an implementation of the Kalman filter. A simple way of using this filter to a stream point data include:

1. Letting the position and the velocity in both axes define the state.
2. Letting the transition matrix be an identity matrix to model a static
3. Using the new cursor points from the feature data as measurements
4. Putting appropriate noise model estimates by making sure that the real noise distribution is not too far from Gaussian white noise assumed by the filter and using the standard deviation obtained when no filters are used to model the noise covariance matrices.

3.2.5 Computer Input / Pointer Behavior

Computer input behavior determines how the pointer actually moves given the final calculation of feature data. The three options investigated for pointer control are called **location**, **velocity**, and **differential** pointers. The differential pointer is used where an interaction zone cannot be defined directly because of lack of available data. Where it is present, it is considered only as a convenience function useful in production software but is not within the scope of this research so it was only implemented in conjunction with some other options. At this point, previous processing components would have provided a pair of values that may be bounded. (e.g. a 2D point with a horizontal and vertical float range of 0-1.0 if bounded)



Determining pointer movement by 1) velocity - calculating the pointer jump from the point with respect to the interaction zone , 2) differential - calculating the pointer jump from the difference between the new and the previous interaction point, and 3) location - calculating the new location from the point with respect to the interaction zone.

Figure 3-8: Pointer Movement Options

Location - The pair of values can determine the actual location of the point on the screen in a straightforward manner by scaling the bounded values to the size of the screen. This option affects accuracy according to the resolution of the screen because of this scaling.

Velocity - With this option, the pair of values will be checked if it is within a certain threshold (i.e. the head is facing generally directly forward). If it is within this threshold, the pointer does not move. After the threshold line, the velocity scales non-linearly from 0 to a pre-determined maximum velocity. This scale is quadratic to allow for finer control while still giving the ability to go as fast as the pre-determined rate. This option allows the head to operate like a joystick and requires less accuracy than the first. Previous works have implemented and recommended this option because of this less stringent accuracy requirement.(Gorodnichy and Roth 2004, 931-942; Varona et al. 2008, 357-374) This joystick-like movement may not be very ideal to users because of the additional movement required to center the face to stop the pointer (as opposed to an actual joystick that centers automatically on its own). Improvements in technology and computer vision techniques allow us to consider and use the first, which can prove to be more usable, as a viable option.

Differential - This option does not require an interaction zone and must be used if the user input style does not provide it in order to follow the design requirement of robustness to user location. Instead of comparing the point to its container, it is compared to the previous point. This difference is then used to calculate how much the pointer will jump. This is comparable to how standard physical mice work. The HeadMouse Extreme, as a PUI-based hardware mouse uses this.

In other input styles, using this option provides a way of calibrating the exact location of the pointer relative to the head pose. For example, the input processor has to know the position and angle of the sensor in order to properly calculate the head pose and location. When differential is used, the user can easily calibrate the pointer location with robustness to reasonable variance in the sensor position and angle.

***Implementation* -** All the abilities in this component are contained in the **`PointerBehavior`** class. The implementation for both the location and velocity options is straightforward. The 2D pair (representing location or rotation) is a vector of floats bounded by 0 and 1. For the velocity option, the interaction point that is bounded by 0 and 1 is shifted to where the origin is placed at the center to make the code that calculates how far the head gaze is from the center position more intuitive. The **`velocityStillRadius`** (ranges from 0-0.5) defines the central region within which the pointer should not be made to move is an adjustable parameter.

With the differential pointer, since there is no interaction zone defined, the interaction point is not scaled to a region bound by 0 and 1 to avoid rounding and arithmetic errors. The previous point is stored in the **`UserInput`** class. The face tracking engine should determine the amount the pointer will jump based on the feature data it makes available (e.g. real-world units, or camera pixels).

3.2.6 Summary of Input Components

The different components in this section (Section 3.2 - Input Components) and the options studied in each are summarized in the following table.

Table 3-1: List of Options and Possible Combinations

USER INPUT			FACE TRACKING (Capture Technology/ Feature Retrieval)				FILTERS (Feature Data Processing)			POINTER (Computer Input)		
Name	Relevant Features	Interaction Zone	MS Face Tracking	FaceAPI	Random Forest	Haar Cascade	None	Average	Kalman	Location	Velocity	Differential
Point from 2D rotation	2D head orientation	Moves based on location. Defined by rotation limits.	o	o	o	x	o	o	o	o	o	o
Point from 2D location	2D point	Fixed. Can move by using Differential pointer.	o	o	o	o	o	o	o	o	x	o
3D Pose Pointing	2D orientation and 3D location	not practically applicable	o	o	o	x	o	o	o	o	o	o
o - possible x - not possible												

There are three (3) investigated options for User Input, four (4) for Face Tracking (which combines options for feature retrieval and capture technology), three (3) for filters, and three (3) for computer input.

3.3 Software Design

I designed the prototype to be able to accomplish its purpose while at the same time allowing easy switching among the different options discussed in Section 3.2 - Input Components. A simplified block diagram of the design follows.

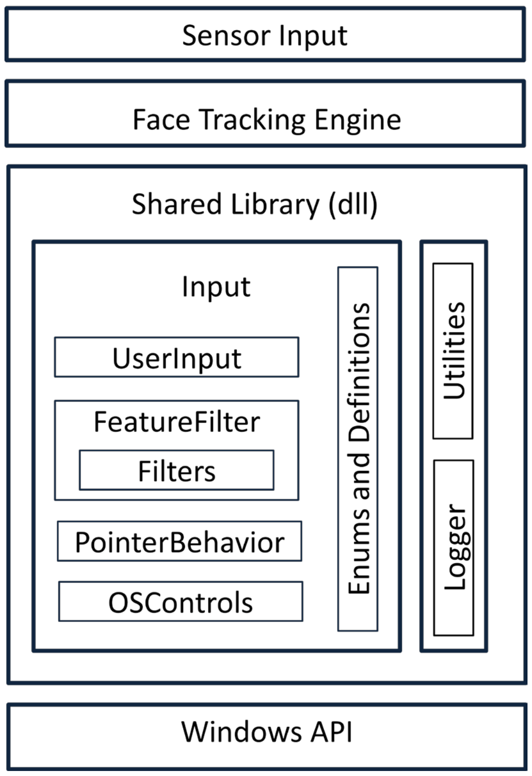


Figure 3-9: Face Tracking User Interface Application Diagram

The applicable input components are written as separate classes - **`UserInput`**, **`FeatureFilter`**, and **`PointerBehavior`**. One main application class (**`Input`**) contains an instance of each of these classes. This application is compiled as a shared library. Different face tracking engines are set up as separate projects and they call methods from the **`Input`** application class. These projects contain an adapter/helper class that, among other things, initializes the `Input` application class with the specified options and makes sure that data is entered in the appropriate format and conventions. While several other classes are contained in the main application class, another such class is **`OSControls`** which sends the commands to the Windows API for input simulation.

The process of getting the final pointer location from the face tracking data is illustrated in the following UML (Unified Modeling Language) sequence diagram.

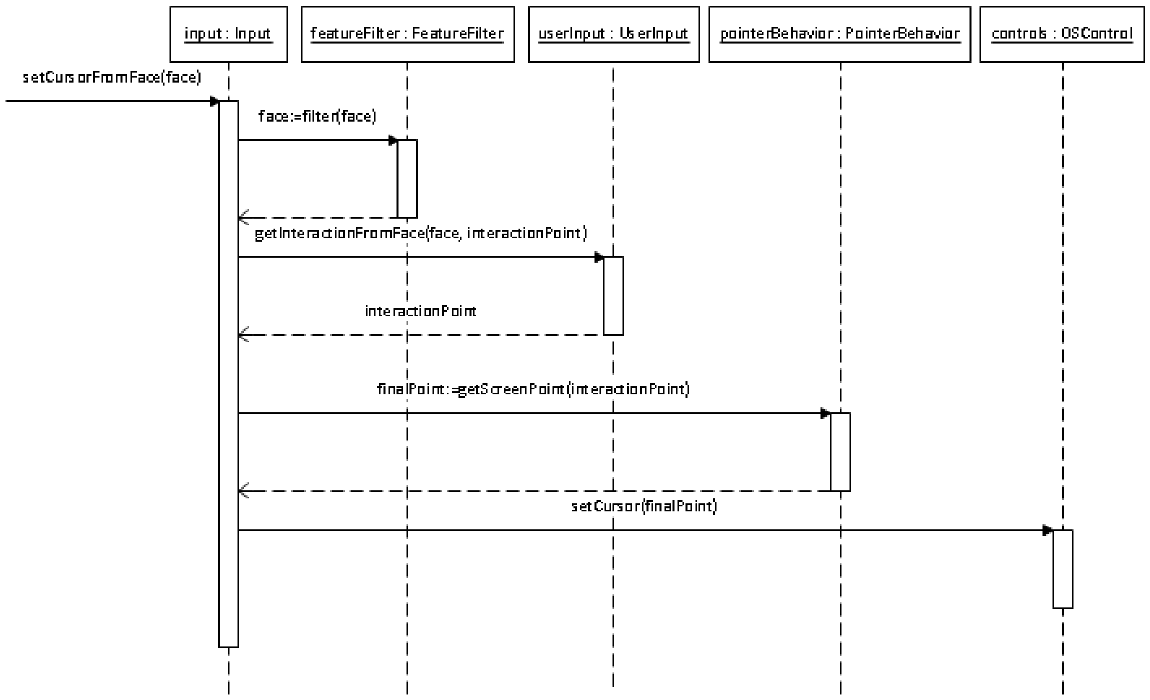

input : Input
featureFilter : FeatureFilter
userInput : UserInput
pointerBehavior : PointerBehavior
controls : OSControl
setCursorFromFace(face)
face:=filter(face)
getInteractionFromFace(face, interactionPoint)
interactionPoint
finalPoint:=getScreenPoint(interactionPoint)
setCursor(finalPoint)

Figure 3-10: Input Data Process (UML Sequence Diagram)

The software design of this prototype is especially suited for the purpose of being able to use and evaluate the different options enumerated in this research. A user-ready product may necessitate a design different from this one.

3.4 Evaluation

I designed and implemented various tests that enable a quantitative comparison of the performance of the different options when used in a face tracking user interface. These evaluation methods are discussed in the following numbered sections. A full usability test is not within the scope of this research and is deferred to future work. The quantitative results should give an indication of good candidates to use for usability tests. Some of the tests measure the speed of usage which is usually a good indication of usability.

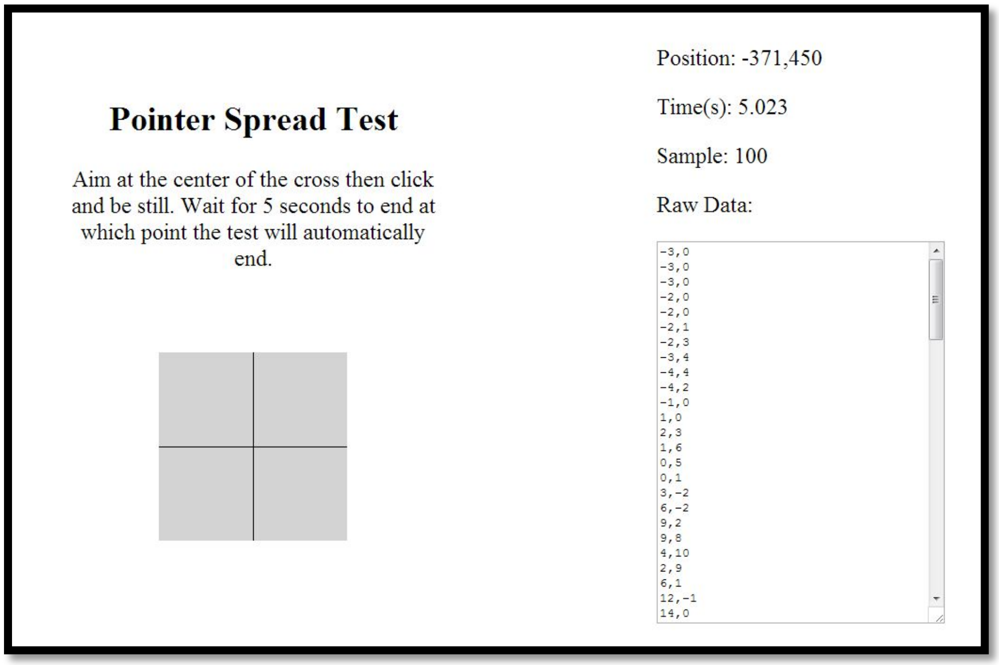
All the tests are implemented as web pages. This allows the tests to be independent to the implementation of the prototypes. This also allows comparison with other solutions that control the pointer because it can be run on any computer with a compliant HTML browser. It is also ideal to have the same user perform all the tests. This removes the most significant factors on user preference and abilities and allows the evaluation of the user interface itself.

The different options also affect the user experience in terms of convenience of usage and setup. These effects were observed and qualitatively discussed.

3.4.1 Point Spread of Stationary Head

One of the tests measures the spread of the generated pointer points of a stationary head over a period of 5 s. This is designed to measure noise and would result in standard deviation values for the horizontal and vertical axes. Setting a time period and an independent data collection cycle length allows for a degree of robustness to possibly differing calculation cycle lengths between implementations (e.g. the length of time required to generate the next point).

With a continuous visual feedback mechanism (i.e. the visible pointer), it is not very crucial to have a specific pose for a particular target pixel. This means that if a particular pose does not make the pointer go to the theoretical position (if one is known or expected), the user can just initiate additional movement to move it to where he or she wants it to go. But as supplemental data, with this test the offset from the target was also calculated.


Pointer Spread Test

Aim at the center of the cross then click and be still. Wait for 5 seconds to end at which point the test will automatically end.

Position: -371,450

Time(s): 5.023

Sample: 100

Raw Data:

```
-3,0
-3,0
-3,0
-2,0
-2,0
-2,1
-2,3
-3,4
-4,4
-4,2
-1,0
1,0
2,3
1,6
0,5
0,1
3,-2
6,-2
9,2
9,8
4,10
2,9
6,1
12,-1
14,0
```

Figure 3-11: Pointer Spread Test

Implementation - Using *Javascript*, the pixel location of the pointer was calculated every 50ms. The test is initiated by a click once the user feels that the pointer is as close to the target as he can make it. The tester will then be still for the next 5 s after which the page will signal the end of the test and the results, containing the standard deviation values and the offset point (calculated from the mean). The raw data vector is also displayed for further analysis.

3.4.2 Speed Pointing

Minimized spread is not enough to evaluate this user interface. The way the pointer follows face movement may be different depending on the implementation. For example, depending on the way filters (see Section 3.2.4 - Feature Data Processing) are implemented, the pointer may exhibit some latency and inertial movement. This would affect the ability to point. For this reason, another test was designed to measure the accuracy of pointing to a target. This is done by measuring the speed at which several points are pointed to and selected. There are nine points placed around and on the center of the screen. Having points in different places would include performance in pointing at locations on the edges of the screen (which potentially can be less accurate than those on the center). The result of this test is a time value.

In the tests done for this research, selection or clicking is initiated using another input device such as a stationary mouse. While it is possible to use dwell or face expressions to initiate it, they were excluded to remove their influence on the result and focus on pointing abilities. It is possible that in trying to click at the target, the user might miss and click outside the target button. The tester is instructed to click when a reasonable aim is made and try to select it as opposed to clicking consecutively and indiscriminately in order to ensuring selection of everything on the path of the pointer.

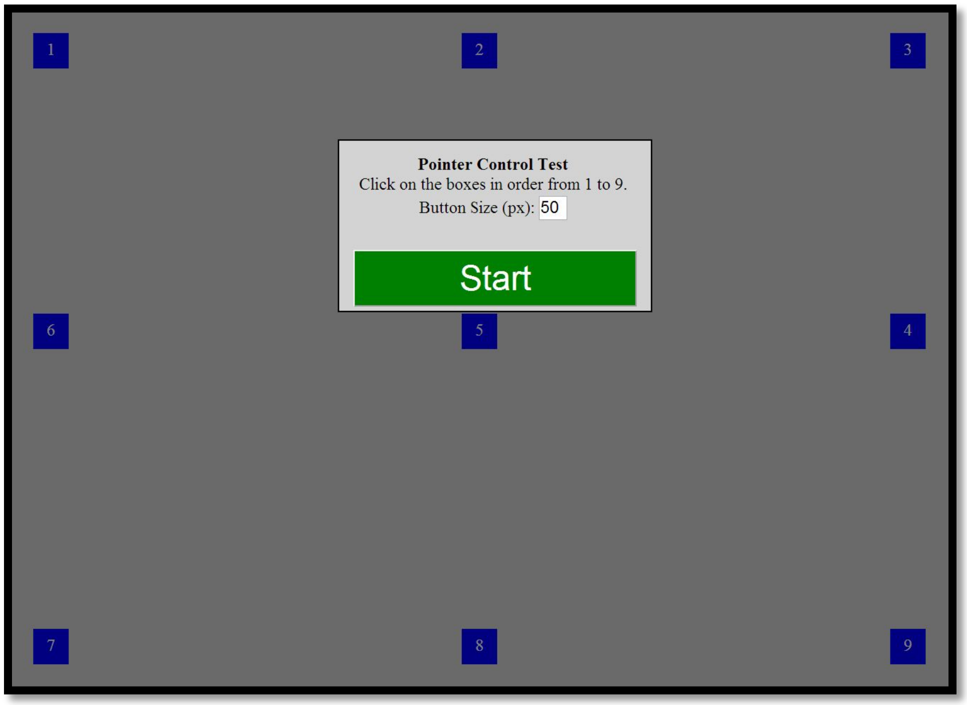

Pointer Control Test
Click on the boxes in order from 1 to 9.
Button Size (px): 50

Figure 3-12: Speed Pointer Test

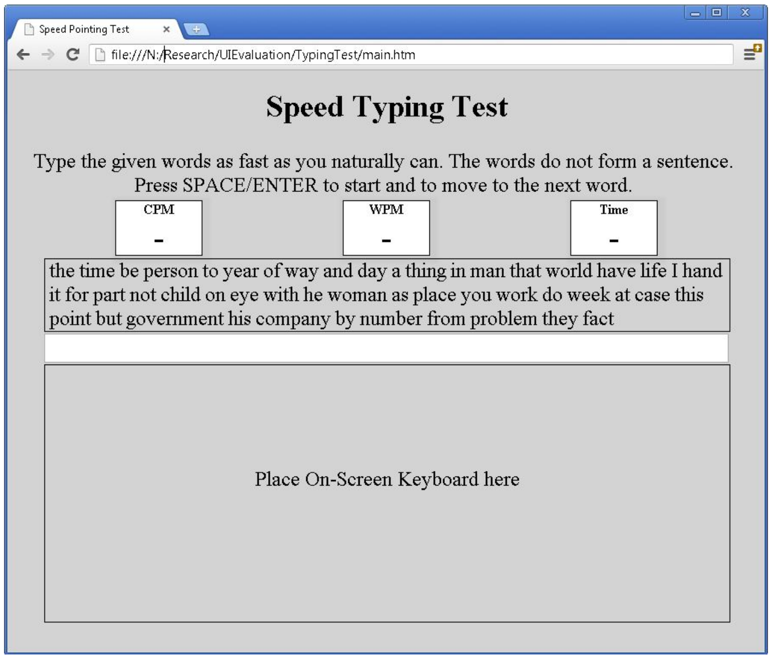
Implementation - The targets are implemented as clickable **`div`**'s. DOM and CSS manipulation is done with the help of *jQuery*. These targets space themselves automatically and use the borders of the browser window. Ideally the test should be conducted with the page at full screen. At the beginning of the test, the targets' size in pixels can be defined. This allows setting the difficulty of the tests. And for simplicity, it is up to the user to make sure that all numbers are selected. Timing starts when the test begins when the designated button is clicked. Timing ends when the last button is clicked. *Javascript's* **`Date().getTime()`** is used which gives millisecond precision.

3.4.3 Speed Typing

Sometimes, missed clicks caused by the challenge to point accurately result in actions that have to be reversed (e.g. the wrong link is selected in an Internet browser). To provide a measurement for the adverse effects of missed clicks, a speed typing test was designed and implemented. In addition, this test will provide comparison among user interfaces that use different modalities (e.g. speech recognition, mouth sticks, etc).

This test presents very common words to be typed using a pointer on an on-screen keyboard. The word list was taken from the top twenty-five most common function words and the top twenty-five most common content words that are nouns in the Oxford English Corpus for a total of fifty words. (Oxford Dictionaries) Since the function words tend to be shorter than the content words, they were placed alternately with the content words. Error correction in this test is forced, which imposes a 100% accuracy rate. This means that every word has to be typed correctly for the test to continue. This accuracy requirement places a time penalty on a missed click because incorrect letters need to be deleted. The test ends when all the words are typed and a minute has not passed or when a minute has passed and the current word is spelled correctly.

A fixed-sized frame (800 x 300 px) is provided as a place holder for the on-screen keyboard. This allows consistency in the size of the keyboard (note that the Windows OSK is resizable) and consequently, in the size of its keys. This makes sure that the size of the keys does not favor one user interface configuration over another. The results are given in characters and words per minute (CPM and WPM). CPM is counted because that measure is invariant to the lengths of the word which, when only a few words are typed, may skew the results. As a final note, no text prediction tool was used (not even the built-in text predictor in the Windows OSK) because key selection is what is being evaluated, not the speed of text entry. Figure 3-12 shows screenshots of the speed typing test before and during a test.



Speed Typing Test

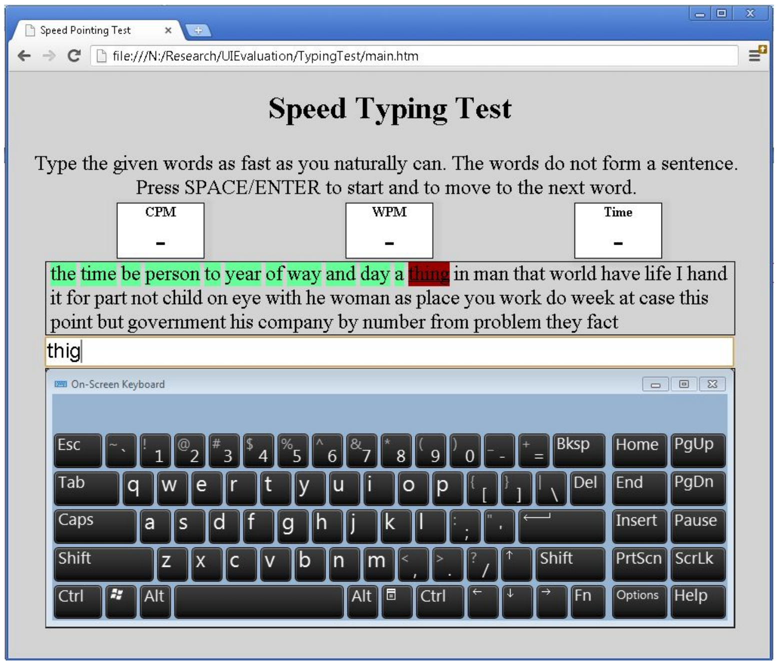
Type the given words as fast as you naturally can. The words do not form a sentence.
Press SPACE/ENTER to start and to move to the next word.

CPM	WPM	Time
-	-	-

the time be person to year of way and day a thing in man that world have life I hand it for part not child on eye with he woman as place you work do week at case this point but government his company by number from problem they fact

Place On-Screen Keyboard here

Figure 3-13: Speed Typing Test - Before Start


Speed Pointing Test

file:///N:/Research/UIEvaluation/TypingTest/main.htm

Speed Typing Test

Type the given words as fast as you naturally can. The words do not form a sentence.
Press SPACE/ENTER to start and to move to the next word.

CPM	WPM	Time
-	-	-

the time be person to year of way and day a thing in man that world have life I hand it for part not child on eye with he woman as place you work do week at case this point but government his company by number from problem they fact

thig

On-Screen Keyboard
Esc ~ ` ! 1 @ 2 # 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 _ - + = Bksp Home PgUp
Tab q w e r t y u i o p { [}] | \ Del End PgDn
Caps a s d f g h j k l : ; " ' Insert Pause
Shift z x c v b n m < , > . ? / ↑ Shift PrtScn ScrLk
Ctrl Alt Alt Ctrl ← ↓ → Fn Options Help

Figure 3-14: Speed Typing Test - Test Ongoing

Implementation - *jQuery* is used for, among other things, its selector capabilities for CSS styling. This makes it easy to implement the formatting for the dynamic status of the words (e.g. correctly spelled, current word, misspelled, etc.) A key event (**`keydown`** and **`keypress`**) handler is attached to the text field to check the status of the current word being typed. The various objects in this client-side application (e.g. **`typingTest`**, **`currentWord`**, and **`testScore`**) are implemented with a modular design pattern that allows these objects to have what are effectively private and public methods.

4 RESULTS AND ANALYSIS

All the options discussed in Section 3.2 - Input Components were implemented. To simplify the evaluation, the different options in a single input component were compared to each other holding all other options constant as much as it is possible instead of testing all 108 combinations, which does not include parameterized options (see Table 3-1: List of Options and Possible Combinations). In addition to quantitative results, some usability considerations are discussed. While only one user was involved in the tests performed, it is sufficient for the purpose as it reduces discrepancies caused by varying human factors. This user was given the opportunity to practice with the different interfaces until he feels he has learned how to use it. For this reason, data from these practice rounds was not collected. Results of this test could be used to direct a full scale usability test of an implementation with some of the best options. After this comparison, an analysis of combined options that could make up a robust implementation is also presented.

Note that in all of the tests conducted, selection or "clicks" were performed by manually clicking a switch (in this case, a stationary mouse). While triggering selection by face gestures (see Section 3.2.1.2 - Selection Control and Other Commands) was implemented for completeness, it was not used in the evaluation because only one of the four face tracking engines used can support it without additional development and/or license fees.

4.1 User Input

There were three options tested for user input styles. They are:

- Point from 2D rotation
- Point from 2D location
- Head pose pointing

The other options were the same in all three conditions (except where specified): namely, MS Face Tracking was used for feature retrieval, simple moving average filter for feature processing, and location for pointer behavior. Note that the filter here was applied uniformly to all interaction points. In later tests, filter processing was modified to work only when the user is within a steadiness threshold.

Because this option directly affects the control of the pointer, the speed pointing test (see Section 3.4.2 - Speed Pointing) was used to evaluate it. For each option, the test was performed 10 times. To provide a comparable baseline, the test was also done using a standard mouse.

Table 4-1: User Input Speed Pointing Results

(sec)	From 2D Location	From 2D Rotation	Head Pose Pointing	Standard Mouse
Average	33.9	43.1	57.8	7.3
Std Dev	3.5	4.7	7.4	0.3
Min	27.4	35.5	47.2	6.9
Max	39.6	49.4	66.7	7.8
Notes: Used MS Face Tracking, average filter, and location pointer From a sequence of 10 tests per option				

Holding all other options and configuration equal and with the same input movement for all three, shorter speeds imply better usability. Longer speeds imply difficulty in pointing at a button and holding still long enough for a click to be completed. Note that missed clicks occurred (though not deliberately as with indiscriminate clicking) but are not considered in this particular test.

From these results, it is clear that using the location of a tracked 2D point ("Point from 2D Location") is the fastest of all three options, although still much slower than the standard mouse. But there is a caveat when it comes to its usability that is a consequence of its implementation. It sets the interaction zone at the beginning of a tracking sequence which then sets the pointer at the center in that frame. If a break happens in the sequence (e.g. tracking fails and the face tracking engine has to restart detection and tracking), or more specifically, if tracking starts while the user's face is in a rotated position, the same position will translate to a pointer being in the center which can be slightly or extremely inaccurate. In this case, tracking may have to be re-initiated by the user. The root cause of this problem is in the face tracking engine which may have varying levels of robustness. It would be ideal if tracking never fails but as it does sometimes, this can be overcome by using a differential pointer (see Section 3.2.5 - Computer Input / Pointer Behavior) or by starting the tracking only when the head is in a neutral position, which can be tricky depending on the data made available by the face tracking engine.

4.2 Face Tracking Engine

Four different face tracking libraries were used in the implementation. They are:

- MS Face Tracking SDK
- SeeingMachine's faceAPI
- Random forest (by Fanelli et al.)
- Haar Cascade Classifier (OpenCV implementation)

The tests for this option include the speed pointing and the measurement of the spread of the generated points of a stationary head (standard deviation). The test pages are implemented externally so there is a common and unbiased test for the different options and their implementation. In this test, point from 2D location was used for user input and location pointer was used for pointer behavior because they are common to all options. In addition, no filtering was applied to show how much noise the face tracking engine itself produces. The computation speeds of each engine were also recorded together with the CPU load. Each of them uses a different visualization tool. It is possible that the visualization of one engine is more intensive than that of the other. No attempts were made to estimate how much computational load visualization causes but this load can be eliminated from user-ready systems because a full resolution visualization of the data stream are not necessary in such

The following table summarizes the results obtained.

Table 4-2: Comparison of Face Tracking Engines

	MS Face Tracking	faceAPI	Random Forest	Haar Cascade
Speed Pointing (s)	22.0	22.3	105.0	>120.0*
Spread - Std Dev (x|y)	10.6 | 9.0	12.7 | 6.3	29.9 | 34.2	77.1 | 40.6
Speed (fps)	24	30	9	16
CPU - Tracking	22%	14%	25%	21%
CPU - Not Tracking	22%	2%	25%	19%
Memory Usage	134MB	99MB	127MB	23MB

Notes:
All implementations (except Random Forest) have a sleep/wait time of 32ms in their main loop.
CPU load is a visually-estimated average from CPU performance counters of MS Windows.
"Point from 2D location", no filters, and location pointer were used.
The standard deviation is an average of the standard deviations of 5 sequences with 100 samples each (5s).
In the Random Forest option, a stride value of 10 was used.
*The subject was unable to complete the speed pointing test

The results show a number of interesting things. MS Face Tracking and faceAPI seem to be similar when it comes to being able to point using location of tracked points. Random Forest and the Haar Cascade are too noisy to be usable. With Random Forest on the speed pointing, indiscriminate clicking helped in completing the task, but in no way can it be considered usable. With Haar Cascade, the task was not completed. Their respective spreads (77.1 and 40.6) show why that is the case.

Their speeds also show which ones are more practical for use. Random Forest maximizes utilization of one of the cores of the quad-core CPU (25% for the entire system) which indicate multi-threaded programming could still improve it but its performance indicate that it needs to improve significantly to be a good consideration for a face tracking user interface. When it comes to speed, faceAPI shows impressive advantage with its lower computational requirement, thereby reaching speeds at the sensor's frame rate. Lower CPU utilization makes this more energy efficient for long-term use.

It is interesting to note the difference in the spread between values in the horizontal (x) and vertical (y) axes. There are several factors that would cause this and much of it is in the implementation of the face tracking engine (e.g. low level of detail in the training data, estimating pose from 2D, etc.). But part of it is also comes from the aspect ratio of the interaction zone (5:4) which was set from the user's level of comfort. This ultimately has to be scaled disproportionately to 16:9 which is the aspect ratio of the screen resolution used for testing.

In the entire development and testing process, observations were made of the performance of the face tracking engine that at this point is more effectively described than quantitatively measured. They are now discussed in turn.

MS Face Tracking works very well in certain conditions but not in some. For one, detection does not work very well when thick glasses are used or when the user has a long, thick beard.(Smolyanskiy 2012) Using the Kinect for Windows sensor (as opposed to the Kinect for Xbox) allows the SDK to detect and track faces that are closer to the sensor than usual (as close as 0.4m) with its near-mode feature. Detection works better when more of the upper skeleton is visible. The skeleton tracking tells the algorithm where to look for the head in the next frame. Without this, continuous detection (as opposed to first time detection) fails. Detection is more robust farther from the sensor, which reduces usability. Tracking is fairly robust to scale and rotation but every now and then a break in the tracking will occur which can be a problem for the user input option "Point from 2D Location" (see Section 4.1 - User Input). The implementation of the options used should be able to recover from such breaks. Overall, this is a good library to use for this user interface but it restricts usage to the Kinect sensor and to the MS platform.

With faceAPI, initial detection requires the user to be within 0.75m of the camera. Detection is robust and there is no noticeable delay. This is another point that gives this advantage over MS Face Tracking. When it comes to detection and tracking, this remains to be a very robust solution. This technology proves that depth is not necessary to have a robust face tracker. In addition, this and the Face tracking SDK are able to detect and track under low light conditions where the primary source of light is the computer monitor. In this lighting scenario, the depth sensor's advantage in seeing in low light conditions is reduced. Unlike the MS Face Tracking SDK, this uses a regular PC camera so hardware costs are minimal. The biggest challenge in its use and deployment is its license restriction.

With MS Face Tracking and faceAPI both being good candidates, the following additional comparison was made for the two. The results that have just been presented did not test their ability to detect head pose (because the option used the 2D location of a tracked point). The following shows speed pointing tests done where the 2D rotation of the head is used as user input.

Table 4-3: Comparison of Head Pose Estimation of MSFT and FaceAPI

Speed Pointing	MSFT	faceAPI
Average (s)	43.1	66.0
Std Dev	4.7	15.9
Min	35.5	45.6
Max	49.4	90.4
Notes: Used average filter, location pointer, and point from 2D rotation		

Though point from 2D rotation is not the best option as the results show (see Section 4.1 - User Input), this comparison shows that the head pose estimation of MS Face Tracking (yaw and pitch, at least) is better than that of faceAPI.

The research of Fanelli et al on random decision forests for head detection and pose estimation from depth using consumer depth images has a published implementation which was used for this prototype.(Fanelli et al. 2011) Because the classifier has been trained on depth images of users from 1m away, classification suffers significantly if the user is not located in the same vicinity. Even with the background segmented out, the presence of other depth patches (e.g. other body parts such as arms and shoulders) could produce false positives and lengthens the computation cycle that is already long (though some parameters can be tweaked for this). The noise, false positives, distance requirement, and long and intensive computation all make this unusable in a face tracking user interface. A stride value can be adjusted that represents the trade-off between accuracy and speed. It has been observed that a more accurate stride level only increases the computation time without significant improvements in accuracy. It has to be noted that compared to other methods, this particular technique has only been recently developed and could still be significantly improved by further research. The same technique has also been recently incorporated into PCL but that particular implementation has not been evaluated for this research.(Aldoma 2012)

In the Haar cascade classifier, tracking is accomplished by successive detection on a per-frame basis, which on its own is quite remarkable. False positives have been observed even in the constrained single user scenario with a flat background although it was uncommon and can be reduced by changing some parameters. Face sizes can vary per frame but for this purpose, it is not needed (though it is used to increase the confidence that the detected head is really the user's head). It is also less robust to face rotation compared to the others so the interaction zone had to be set smaller than the normal. All of these make it insufficient on its own to enable a face tracking user interface.(Villaroman and Rowe 2012, 57-62)

4.3 Filters

The following filters were evaluated:

- Average
- Kalman
- None used

The purpose of using filters is to help the user control the pointer more accurately. To evaluate these options, two things were looked into - 1) how does it minimize the noise when pointing at a target and 2) how does it affect the actual pointing movement. It is possible to configure the filter in such a way that it minimizes noise to the detriment of usability. For example, increasing the sample size of a moving average filter may result in a minimal noise but it may make the pointer move with significant latency. For this reason, the filtering was only applied when the face is within a steadiness threshold.

To answer these questions, the point spread test and the speed pointing tests were used. MS face tracking, "Point from 2D Location", and a location pointer were the common options. Figure 4-1 shows a histogram of the generated points (using the point spread test) comparing the different filter options. Table 4-4 shows a summary of the results of both the point spread test and the speed pointing test.

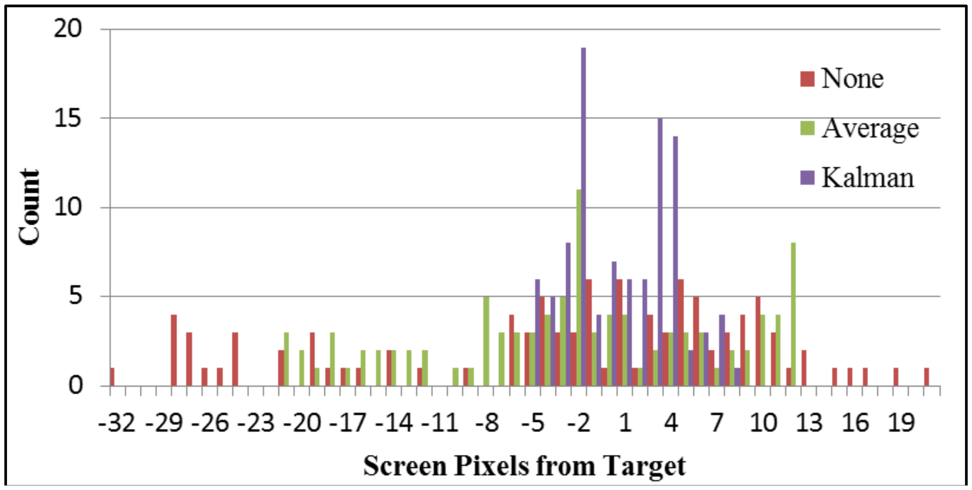


Figure 4-1: Comparison of Filter Performance from a Sample of Generated Points

Table 4-4: Comparison of Feature Filters

	None	Average	Kalman
Point spread (x|y)	10.6 | 8.9	7.9 | 3.4	3.5 | 4.5
Speed pointing (s)	22.0	33.9	22.1

Notes:
Used MS Face Tracking, point from 2D location, and location pointer
Point spread is the average of the standard deviations of five test sequences
Speed pointing is the result of 10 test sequences

Some of the results here were unexpected. It may be intuitive to assume that applying filters would make effective use easy. While the point spread did decrease from having none to using average and from then to using the Kalman filter signifying less noise, the speed pointing test results show that it wasn't necessarily better. In fact the average filter performed poorly when it comes to effective use. Just as discussed earlier, smoothing has unintended adverse consequences. It makes the pointer move with sufficient latency that the user would find himself trying to click while crossing the mark which he might then miss because of the size of the button and the speed of the momentum of the pointer. At the same time, while having no filter seemed like it worked effectively, which it did, the problem with it is not very evident here. Because the pointer will jump around in the area, targeting smaller objects becomes more of a challenge. It just so happened that the 50-pixel buttons on the test was big enough that the random pointing was likely to hit the button. When the Kalman filter was first used, while it did solve the lag problem, it proved to be less usable than expected because, as a predicting algorithm, it causes an inertial effect, where the pointer can go beyond the desired point based on the previous direction. It also causes a "boomerang" effect as the pointer goes back to the desired point. These inertial effects were very evident with broad face movements.

To solve this problem, the filter was implemented to work only when helpful. The movement of pointing to a screen target can be divided into two phases - the relocation phase, where the pointer is moved to the general area of the target and which is usually done with broad swift movements, and the targeting phase, where the pointer is moved to the actual target and which is usually done in a slower manner. The solution used to solve this problem is to apply the filter only in the targeting phase but not in the relocation phase. This has improved the performance of this filter to figures presented (see Table 4-4: Comparison of Feature Filters).

4.4 Computer Input/Pointer Behavior

The three options evaluated were:

- Location - point location is controlled and is determined by scaled bounded interaction point
- Velocity - point velocity is controlled and is operated like a joystick
- Differential - point jump is controlled by the difference of previous and current interaction point and is operated like a mouse that cannot be lifted.

While the differential pointer is mentioned here, its performance is the same as the location pointer because the math behind both is the same. If there is any difference between the two it is artificial and it will come from the difference in the sizes of the interaction zone which is inherently challenging to make the same (one is from the change in a face movement and the other from the limit of the face movement). The differential pointer just has the advantage of convenient self-calibration of the pointer offset and provides some robustness to variance in sensor's location and orientation. Because the performance of the differential and the location pointer is the same, only the location and the velocity pointers are compared in this section. The speed pointing test is used.

Table 4-5: Comparison of Pointer Behavior Options

Speed Pointing	Location	Velocity
Average (s)	29.03	49.38
Std Dev	2.88	6.10
Min	23.43	37.54
Max	32.73	59.70
Notes: Used faceAPI, average filter, and point from 2D location The still radius for velocity (where the pointer will not move) is 30% of the interaction zone and the maximum pixel change is 15 per computation cycle.		

This shows that from the same interaction point space, setting the pointer by location is faster than when setting the velocity is used. The face movement of the former is more natural while that of the latter is more erratic because the face has to go back to neutral position to stop. While it helped that the velocity changes in a non-linear way, it still made it challenging to make fine adjustments to the pointer. While one can get used to its settings over time as evidenced by worse results in the very beginning, the location pointer still triumphs when it comes to naturally learning the interaction style.

4.5 Summary of Results

In Section 1.5 - Research Hypotheses, a list of hypotheses was presented. The following summarizes the results of the experiments done with them.

Table 4-6: Results by Hypotheses

	Hypotheses	Result
1a	State-of-the-art pose estimation makes pointing more accurate	False
1b	State-of-the-art pose estimation is sufficient for head pose pointing (user input option)	False
2a	Depth data makes detection and tracking of face features more robust	True
2b	Depth data is necessary to make detection and tracking of face features more robust	False
3	Tracking is better than rapid detection	True
4	Current face tracking tech is able to support a robust face tracking UI	Sometimes
5	The Kalman filter deals with noise better than an average filter	True
6	Pointing operation is easier when noise is minimized	Sometimes
7	A Location/differential pointer is easier to use than a velocity one	True
8	A differential pointer reduces the data requirements on the face tracking engine	True
9	External pointing tests can be used to compare performance	True
10	Pointing accuracy can be measured by the stationary point spread	True

In addition, the following summarizes some of the more significant discoveries and contributions of this research when it comes to the design and implementation of face tracking user interfaces.

This research provides a logical framework for the design, implementation, and evaluation of face tracking user interfaces. Previous work only focused on one or two components, and implemented a reasonable option from the other components to complete it.

This research lays out a more complete picture of all the different components of this kind of user interface so future work can use it to focus on a component in the appropriate context.

The interaction point should come from the space that has more detail and larger span. This was evident in the increased effectiveness of tracking a point from 2D location over doing so from 2D rotation. As far as the face tracking engines reviewed here have implemented head pose estimation, there is less noise and more detail in the camera pixel space than in orientation estimates. The reviewed state-of-the-art techniques on head pose estimation did not prove to be necessary or very helpful in coming up with the most robust solution from the options. The span of possible values with which to interpret input is defined by setting interaction zones which is determined by the comfortable physical limits of the user and is implemented virtually.

Depth data is not necessary in making tracking sufficiently robust for a face tracking user interface, though it may help improve the performance in some known and novel techniques. faceAPI has shown this in that even though it does not empirically measure depth data, it still performed as well or better than some options that do. faceAPI has been verified to be a high standard for face tracking. Though MS Face Tracking was shown to be better than faceAPI with pose estimation due to the availability of real depth, they are on the same level in their ability to track face location even while one uses depth and other does not. Depth measurements from consumer devices such as the Kinect are noisy and have to be processed further and in novel ways to be able to track fine face movement. However, using the color image and depth image together can make face tracking more robust as shown by MS Face Tracking.

From capturing raw data at the beginning to the final cursor location at the end of the input process, the coordinate spaces of the 2D/3D data have to be clearly defined. In this research, an understanding and application of this concept allowed different face tracking engines to be used with a common input processing library and to be switched easily.

It was shown that filters can have both positive and negative effects on the usability of a face tracking user interface. There can be a trade-off between a filter's ability to reduce noise and its effect on usability. While an appropriately configured Kalman filter (or other filters) can reduce noise effectively, the goal of being able to aim accurately should be kept in mind. The filters have to be implemented in such a way as to avoid pointing problems as a direct result of the filter. Some of these filter-induced problems include lag and inertia. These problems were addressed here in what is believed to be a novel way by using a filter and implementing it in such a way that it adapts to new data points quickly and is only applied when the user is in targeting mode (as opposed to broad movements of the face to move the pointer to a general area).

Controlling the velocity of a pointer was shown to be less effective or slower than controlling the actual location. In addition, using the face like a joystick involves more movement which can be felt as unnatural. The differential pointer is a different implementation approach to setting the interaction zone and calibration. This solves practical problems such as incorrect settings and calibration that can be caused by, among other things, the sensor's location and orientation and the accuracy of the coordinate spaces of the face tracking engine.

From the tests performed, the following combination, hereafter called "faceUI", produced the best result.

Table 4-7: Best Possible Combination from Results

INPUT COMPONENT	OPTION
User Input	Point from 2D Location
Face Tracking Engine	faceAPI
Feature Filter	Kalman Filter
Pointer Behavior	Location/Differential Pointer

Some of the latest existing face-tracking user interfaces were compared with faceUI. Using these other solutions, the speed pointing and the point spread tests were performed. The following table summarizes the results obtained.

Table 4-8: Comparison with Existing Solutions

Test		faceUI	Head Mouse	Smart Nav	Camera Mouse	Standard Mouse
Speed Pointing (s)	Avg	19.2	14.9	18.3	20.8	8.1
	σ	1.8	1.2	1.0	1.5	0.7
	Min	16.8	13.5	16.8	18.2	7.4
	Max	22.7	17.0	19.9	22.6	9.7
Spread, σ (x|y)		3.2 | 3.1	1.6 | 1.2	2.8 | 1.9	6.7 | 3.0	0.0 | 0.0
Speed Typing	**cpm**	39.6	53.4	52.1	25.8	98
	wpm	9.6	13.2	12.8	6.0	23.4

This shows that faceUI is comparable with the other solutions in performance though certainly not the best. This shows great potential in this particular combination but with additional usability benefits as defined in Section 1.2 - Proposed User Interface. However, the biggest disadvantage of this combination is the proprietary nature and commercial licensing attached to faceAPI. The HeadMouse Extreme, thought by the staff of the Utah Center for Assistive Technology as the best option available in terms of performance, value, and ease of installation and use, remains as the best performing solution among those that were evaluated in this thesis (next to the standard mouse).

Because the same tests were used, these different user interface solutions can be compared with confidence that the results are indicative of performance. Part of the reason the speed typing test was created and used is to allow comparison with other solutions that may use different modalities. But there are some caveats in that the experimental setup may be different. For example, Dasher, a predictive text entry tool evaluated with an eye gaze tracking system was able to produce typing results ranging from 2.5-17.3 wpm (range from first to tenth sessions). (Tuisku et al. 2008, 19-26) While the higher of these results outperform many of the leading solutions evaluated in my research when it comes to nominal wpm, the results are not strictly comparable because it uses word prediction while the speed typing test that I conducted was designed not to use any prediction (because it is the key press accuracy that is being measured). The same thing can be said of text entry speed ranging from 7.6 wpm to 86.9 wpm in a survey that gathered results from experiments that used different text entry methods, keyboard configurations, and error correction conditions. (Arif and Stuerzlinger 2009, 100-105) But there is still value in knowing the typical typing speeds from these face tracking user interfaces because they are, after all, actual typing speeds. Where speech recognition is an available modality, it can yield significantly higher typing speeds (e.g. 160 wpm) and may be best for text entry. (M2PressWIRE 2008)

Finally, a significant result of this research is that tests were designed and implemented to measure the performance of the differently configured face tracking user interfaces. They provided a standard of measurement for this research and are proposed to be a standard for other input systems that rely on simulating pointer control.

5 CONCLUSION AND FUTURE WORK

Face tracking user interfaces can provide significant benefit for users with certain disabilities or who otherwise do not have full control of their hands but have sufficient control of their heads. In this work, the entire input process of face tracking user interfaces was divided into logical and functional components namely, user input, capture technology, feature retrieval, feature processing/filtering, and pointer behavior. Different options were evaluated for each component and considerations for their selection were given. These options were implemented and their performance in a user interface was quantitatively measured. Some qualitative evaluation on their usability as far as they have been tested and observed was also made.

Among other results of this research are the following findings. It was determined that the SeeingMachines faceAPI library and the Microsoft Face Tracking SDK are leading face tracking engines that could enable a face tracking user interface. Depth data from depth-capable sensors may be helpful but is not necessary for good performance. Location-based tracking is still better to use than orientation-based tracking. Filters have to be implemented to improve the ability of users to point by both minimizing the noise from the face tracking engine and by minimizing consequential lag and inertia.

Other challenges of designing and implementing face tracking user interfaces using consumer devices were also discussed. These challenges together with design and usability requirements necessitate a sufficiently high level of accuracy and robustness to the face tracking engine and the processing of its output.

The tests that were used also provided a good method of comparison of performance of face tracking user interfaces. Because these tests can be performed in several major Internet browsers, it is planned to have these tests be published on the Internet as a web application for use as a standard for other user interfaces that are designed to control a computer pointer.

This work can be expanded or used in several ways. First, since all the tests here were done by a single user, it is possible that certain options may be more suitable for some users than others. For example, with a particular face tracking engine, detection and tracking may work better for one user than for another because of the features of the user's face. Or using velocity pointers may be easier (e.g. possibly for some users with some spasticity in their face movement). In any case, it would be important to have the options presented here or combinations of them be evaluated in a full-scale usability test. Doing so may help uncover issues that might be present but are not easily detectable in the single user tests that were done (e.g. possible learnability issues).

There is also significant work that can be done on the individual components. While there are many, some of these include:

- Identification, implementation, and evaluation of techniques that can accomplish face detection and tracking in video sequences that is comparable to or exceeds the performance of faceAPI / MS Face Tracking and which will be made available to the open-source community
- Investigation of visual feedback techniques that will help with usability which was not in the scope of this research

- Evaluation of the feasibility, implementation, and usability of triggering special commands using face gestures which was discussed here and even implemented for completeness but was not evaluated.

As computing theory and technology advances, work should be done in using them to provide valuable services to the people. These services can be more personally significant to individuals who have to deal with disadvantages and disabilities in life. It has been the aim of this research to investigate and solve technical problems in helping individuals be able to use their heads (by necessity or preference) to control a personal computer and this has been accomplished to a degree although certainly, much more can be done.

REFERENCES

Adafruit Industries, "The Open Kinect Project – the Ok Prize – Get $3,000 Bounty for Kinect for Xbox 360 Open Source Drivers" http://www.adafruit.com/blog/2010/11/04/the-open-kinect-project-the-ok-prize-get-1000-bounty-for-kinect-for-xbox-360-open-source-drivers/ (accessed May 2012 2012).

Aldoma, A. "Progress on Head Detection and Pose Estimation (Ii)." In *PCL Developers blog*, 2012, 2012.

Arif, A. S., and W. Stuerzlinger. "Analysis of Text Entry Performance Metrics." In *Science and Technology for Humanity (TIC-STH), 2009 IEEE Toronto International Conference*, 100-105, 2009.

Balasubramanian, V. N., Y. Jieping, and S. Panchanathan. "Biased Manifold Embedding: A Framework for Person-Independent Head Pose Estimation." In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, 1-7, 2007.

Besl, P. J., and H. D. McKay. "A Method for Registration of 3-D Shapes." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 14, no. 2 (1992): 239-256.

Betke, M., J. Gips, and P. Fleming. "The Camera Mouse: Visual Tracking of Body Features to Provide Computer Access for People with Severe Disabilities." *Neural Systems and Rehabilitation Engineering, IEEE Transactions on* 10, no. 1 (2002): 1-10.

Bhandarkar, S. M., and X. Luo. "Integrated Detection and Tracking of Multiple Faces Using Particle Filtering and Optical Flow-Based Elastic Matching." *Computer Vision and Image Understanding* 113, no. 6 (2009): 708-725.

Bilmes, J. A., X. Li, J. Malkin, K. Kilanski, R. Wright, K. Kirchhoff, A. Subramanya, S. Harada, J. A. Landay, P. Dowden, and H. Chizeck. "The Vocal Joystick: A Voice-Based Human-Computer Interface for Individuals with Motor Impairments." In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, 995-1002. Vancouver, British Columbia, Canada: Association for Computational Linguistics, 2005.

Chathuranga, S. K., K. C. Samarawickrama, H. M. L. Chandima, K. G. T. D. Chathuranga, and A. M. H. S. Abeykoon. "Hands Free Interface for Human Computer Interaction." In *Information and Automation for Sustainability (ICIAFs), 2010 5th International Conference on*, 359-364, 2010.

Dementhon, D. F., and L. S. Davis. "Model-Based Object Pose in 25 Lines of Code." *Int. J. Comput. Vision* 15, no. 1-2 (1995): 123-141.

Di, H., S. Caifeng, M. Ardabilian, W. Yunhong, and C. Liming. "Local Binary Patterns and Its Application to Facial Image Analysis: A Survey." *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 41, no. 6 (2011): 765-781.

Fagiani, C., M. Betke, and J. Gips. "Evaluation of Tracking Methods for Human-Computer Interaction." In *Applications of Computer Vision, 2002. (WACV 2002). Proceedings. Sixth IEEE Workshop on*, 121-126, 2002.

Fanelli, G., J. Gall, and L. V. Gool. "Real Time 3d Head Pose Estimation: Recent Achievements and Future Challenges." In *Communications, Control and Signal Processing, 2012. ISCCSP 2012. 5th International Symposium on*, 2012.

Fanelli, G., T. Weise, J. Gall, and L. V. Gool. "Real Time Head Pose Estimation from Consumer Depth Cameras." In *DAGM'11*. Frankfurt, Germany, 2011.

Froba, B., and A. Ernst. "Face Detection with the Modified Census Transform." In *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, 91-96, 2004.

Gips, J., and P. Olivieri. "Eagleeyes: An Eye Control System for Persons with Disabilities." In *Eleventh International Conference on Technology and Persons with Disabilities*. Los Angeles, 1996.

Gorodnichy, D. *Perceptual Cursor - a Solution to the Broken Loop Problem in Vision-Based Hands-Free Computer Control Devices*. 2006.

Gorodnichy, D. O. "On Importance of Nose for Face Tracking." In *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*, 181-186, 2002.

Gorodnichy, D. O., and G. Roth. "Nouse 'Use Your Nose as a Mouse' Perceptual Vision Technology for Hands-Free Games and Interfaces." *Image and Vision Computing* 22, no. 12 (2004): 931-942.

gwr_press. "Kinect Confirmed as Fastest-Selling Consumer Electronics Device." In *Guinness World Records*, 2012, 2011.

Heikkil, H., #228, K.-J. R, and ih. "Simple Gaze Gestures and the Closure of the Eyes as an Interaction Technique." In *Proceedings of the Symposium on Eye Tracking Research and Applications*, 147-154. Santa Barbara, California: ACM, 2012.

Hongliang, J., L. Qingshan, L. Hanqing, and T. Xiaofeng. "Face Detection Using Improved Lbp under Bayesian Framework." In *Image and Graphics, 2004. Proceedings. Third International Conference on*, 306-309, 2004.

Hunke, M., and A. Waibel. "Face Locating and Tracking for Human-Computer Interaction." In *Signals, Systems and Computers, 1994. 1994 Conference Record of the Twenty-Eighth Asilomar Conference on*, 2, 1277-1281 vol.2, 1994.

Inhyuk, M., K. Kyunghoon, R. Jeicheong, and M. Museong. "Face Direction-Based Human-Computer Interface Using Image Observation and Emg Signal for the Disabled." In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, 1, 1515-1520 vol.1, 2003.

Instruments, O., "Headmouse® Extreme: Wireless Head Contolled Mouse" http://www.orin.com/access/headmouse/ (accessed Nov 2012).

Istance, H., R. Bates, A. Hyrskykari, and S. Vickers. "Snap Clutch, a Moded Approach to Solving the Midas Touch Problem." In *Proceedings of the Eye Tracking Research and Applications Symposium 2008*, 221-228: ACM, 2008.

Jacob, R. J. K. "The Use of Eye Movements in Human-Computer Interaction Techniques: What You Look at Is What You Get." *ACM Trans. Inf. Syst.* 9, no. 2 (1991): 152-169.

Jörgen, A. "Candide-3 - an Updated Parameterised Face." (2001).

Khoshelham, K., and S. O. Elberink. "Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications." *Sensors* 12, no. 2 (2012): 1437-1454.

Knies, R. "Academics, Enthusiasts to Get Kinect Sdk." 2012: Microsoft Research, 2011.

Kondori, F. A., S. Yousefi, L. Haibo, and S. Sonning. "3d Head Pose Estimation Using the Kinect." In *Wireless Communications and Signal Processing (WCSP), 2011 International Conference on*, 1-4, 2011.

Küblbeck, C., and A. Ernst. "Face Detection and Tracking in Video Sequences Using the Modifiedcensus Transformation." *Image and Vision Computing* 24, no. 6 (2006): 564-572.

M2PressWIRE. "Nuance Dragon Naturallyspeaking 10 - Speech Recognition That Changes People's Lives; Nuance Announces Winners of the "I Speak Dragon" User Story Contest; New Release Hailed by Users and Experts Alike." M2PressWIRE, 2008.

Majaranta, P., U.-K. Ahola, and O. Špakov. "Fast Gaze Typing with an Adjustable Dwell Time." In *Proceedings of the SIGCHI conference on Human factors in computing systems*, 357-360. Boston, MA, USA: ACM, 2009.

Microsoft. "Kinect for Windows Human Interface Guidelines V1.5.0." (2012). http://www.microsoft.com/en-us/kinectforwindows/develop/learn.aspx [accessed May 2012].

Ming-Hsuan, Y., D. J. Kriegman, and N. Ahuja. "Detecting Faces in Images: A Survey." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 24, no. 1 (2002): 34-58.

Mingcai, Z., L. Lin, S. Jian, and W. Yangsheng. "Aam Based Face Tracking with Temporal Matching and Face Segmentation." In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 701-708, 2010.

Morris, T., and V. Chauhan. "Facial Feature Tracking for Cursor Control." *Journal of Network and Computer Applications* 29, no. 1 (2006): 62-80.

MSDN, "Coordinate Spaces" http://msdn.microsoft.com/en-us/library/hh973078.aspx (accessed Nov 2012).

Muchun, S., Y. Chinyen, L. Shihchieh, W. Pachun, and H. Shawmin. "An Implementation of an Eye-Blink-Based Communication Aid for People with Severe Disabilities." In *Audio, Language and Image Processing, 2008. ICALIP 2008. International Conference on*, 351-356, 2008.

Murphy-Chutorian, E., and M. M. Trivedi. "Head Pose Estimation in Computer Vision: A Survey." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 31, no. 4 (2009): 607-626.

Oxford Dictionaries, "The Oec: Facts About the Language" http://oxforddictionaries.com/words/the-oec-facts-about-the-language (accessed Jan 2013).

Pan, H., Y. Zhu, and L. Xia. "Efficient and Accurate Face Detection Using Heterogeneous Feature Descriptors and Feature Selection." *Computer Vision and Image Understanding* 117, no. 1 (2013): 12-28.

Point, N., "Smartnav" http://www.naturalpoint.com/smartnav/ (accessed Nov 2012).

Prasad, B. H. P., and R. Aravind. "A Robust Head Pose Estimation System for Uncalibrated Monocular Videos." In *Proceedings of the Seventh Indian Conference on Computer Vision, Graphics and Image Processing*, 162-169. Chennai, India: ACM, 2010.

PrimeSense, "The Primesense 3d Awareness Sensor", PrimeSense Ltd http://primesense.com/press-room/resources/file/4-primesense-3d-sensor-data-sheet?lang=en (accessed May 2012 2012).

Reale, M., T. Hung, and Y. Lijun. "Pointing with the Eyes: Gaze Estimation Using a Static/Active Camera System and 3d Iris Disk Model." In *Multimedia and Expo (ICME), 2010 IEEE International Conference on*, 280-285, 2010.

Rusu, R. B., and S. Cousins. "3d Is Here: Point Cloud Library (Pcl)." In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 1-4, 2011.

Shotton, J., A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. "Real-Time Human Pose Recognition in Parts from Single Depth Images." In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, 1297-1304, 2011.

Smolyanskiy, N. "Face Tracking Sdk in Kinect for Windows 1.5." In *My Random List*, 2012, 2012.

Toan Thanh, D., D. Khiem Ngoc, L. Thai Hoang, and L. Bac Hoai. "Boosted of Haar-Like Features and Local Binary Pattern Based Face Detection." In *Computing and Communication Technologies, 2009. RIVF '09. International Conference on*, 1-8, 2009.

Tu, J., H. Tao, and T. Huang. "Face as Mouse through Visual Face Tracking." *Computer Vision and Image Understanding* 108, no. 1–2 (2007): 35-40.

Tuisku, O., P. Majaranta, P. Isokoski, and K.-J. Räihä. "Now Dasher! Dash Away!: Longitudinal Study of Fast Text Entry by Eye Gaze." In *Proceedings of the 2008 symposium on Eye tracking research &#38; applications*, 19-26. Savannah, Georgia: ACM, 2008.

Varona, J., C. Manresa-Yee, and F. J. Perales. "Hands-Free Vision-Based Interface for Computer Accessibility." *Journal of Network and Computer Applications* 31, no. 4 (2008): 357-374.

Vazquez, L. J. G., M. A. Minor, and A. J. H. Sossa. "Low Cost Human Computer Interface Voluntary Eye Movement as Communication System for Disabled People with Limited Movements." In *Health Care Exchanges (PAHCE), 2011 Pan American*, 165-170, 2011.

Villaroman, N., D. Rowe, and B. Swan. "Teaching Natural User Interaction Using Openni and the Microsoft Kinect Sensor." In *Proceedings of the 2011 conference on Information technology education*, 227-232. West Point, New York, USA: ACM, 2011.

Villaroman, N. H. "Face Tracking User Interfaces Using Vision-Based Consumer Devices." In *Proceedings of the 14th international ACM SIGACCESS conference on Computers and accessibility*, 297-298. Boulder, Colorado, USA: ACM, 2012.

Villaroman, N. H., and D. C. Rowe. "Improving Accuracy in Face Tracking User Interfaces Using Consumer Devices." In *Proceedings of the 1st Annual conference on Research in information technology*, 57-62. Calgary, Alberta, Canada: ACM, 2012.

Viola, P., and M. Jones. "Rapid Object Detection Using a Boosted Cascade of Simple Features." In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, 1, I-511-I-518 vol.1, 2001.

Wang, J.-G., and E. Sung. "Em Enhancement of 3d Head Pose Estimated by Point at Infinity." *Image and Vision Computing* 25, no. 12 (2007): 1864-1874.

WebAIM, "Motor Disabilities: Assistive Technologies" http://webaim.org/articles/motor/assistive (accessed July 2012 2012).

Weise, T., S. Bouaziz, H. Li, and M. Pauly. "Realtime Performance-Based Facial Animation." *ACM Trans. Graph.* 30, no. 4 (2011): 1-10.

Xiaogang, Z., and H. Yanbo. "Face Tracking Based on Fusion Skin Color Model and Optical Flow Algorithm." In *Wireless Networks and Information Systems, 2009. WNIS '09. International Conference on*, 89-92, 2009.

Yagi, T. "Eye-Gaze Interfaces Using Electro-Oculography (Eog)." In *Proceedings of the 2010 workshop on Eye gaze in intelligent human machine interaction*, 28-32. Hong Kong, China: ACM, 2010.

Yilmaz, A., O. Javed, and M. Shah. "Object Tracking: A Survey." *ACM Comput. Surv.* 38, no. 4 (2006): 13.

Yun, F., and T. S. Huang. "Graph Embedded Analysis for Head Pose Estimation." In *Automatic Face and Gesture Recognition, 2006. FGR 2006. 7th International Conference on*, 6 pp.-8, 2006.

Zhang, C., and Z. Zhang. "A Survey of Recent Advances in Face Detection." *Microsoft Research*, no. MSR-TR-2010-66 (2010). http://research.microsoft.com/apps/pubs/default.aspx?id=132077 [accessed June 2012].

Zheng, W., and S. M. Bhandarkar. "Face Detection and Tracking Using a Boosted Adaptive Particle Filter." *Journal of Visual Communication and Image Representation* 20, no. 1 (2009): 9-27.