Improving Accuracy in Face Tracking
User Interfaces using Consumer Devices

Norman Villaroman
Brigham Young University
Provo, Utah
normanhv@byu.edu

Dale Rowe
Brigham Young University
Provo, Utah
dale_rowe@byu.edu

ABSTRACT
One form of natural user interaction with a personal computer is based on face pose and location. This is especially helpful for users cannot effectively use common input devices with their hands. A characteristic problem of such an interface, among others, is that face movement is expected to be small and limited relative to a significantly larger control area (e.g. a full resolution monitor). In addition, vision-based algorithms and technologies that enable such interfaces introduce noise that adversely affects usability. This paper describes some of these problems in detail and presents potential solutions. Some basic face tracking user interfaces with different configurations were implemented and statistically evaluated to support the analysis. The different configurations include the use of 2D and depth images (from consumer depth sensors), different input styles, and the use of the Kalman filter.

Categories and Subject Descriptors
H.5.2 [Information Interfaces and Presentation]: User Interfaces – Input devices and strategies.
I.4.8 [Image Processing and Computer Vision]: Scene Analysis – Object recognition, Tracking.
I.5.5 [Image Processing and Computer Vision]: Implementation – Interactive systems.

General Terms
Algorithms, Human Factors

Keywords
Hands-Free, Face Tracking, Detection, User Interface, Accessibility, Depth, Consumer Devices

1. INTRODUCTION
As technology permeates modern society, various forms of user interfaces (UI) have become more prevalent. Keypads or other similar input devices with physical buttons are replaced or supplemented by devices such as touch pads, speech recognition, accelerometers, and imaging devices, to name a few.

People with certain disabilities may find common interfaces very restrictive. As technology advances and as awareness of various user needs grows, accessibility and assistive technology continue to establish themselves as integral parts of the field of information technology.[1, 2] An extended set of technology and user interfaces are required for those who find great difficulty, if not a total impossibility, to use and manipulate hand-held devices such as the ubiquitous mouse and keyboard. Users in this group can be further divided into their other abilities or lack thereof (e.g. sight, speech, motor skills, cognitive abilities, etc). Various solutions exist for these groups. The user interface referred to in this paper is one in which users can provide cursor and selection control to a computer just by using natural head and face movements without attached or intrusive devices.

We mention first some past and recent works on hands-free and vision-based user interfaces. Then, the input data flow is divided into different components and the accuracy problem is discussed for each one. Basic prototypes are then presented that serve to show how different design options affect accuracy, and of which only a statistical analysis is made. A usability study is deferred for future work.

2. RELATED WORK
The need for user interfaces that cater to individuals who cannot use traditional hand-held input devices have long been known, with and can be found in several works in the 80's and 90's [3, 4]. As computer vision advances, its algorithms helped improved hands-free user interfaces. Sapaico and Sato detected tongue protrusions and translated them into Morse code for text input.[5] Muchun et al detected eye blinks using pattern matching and optical flow.[6] The Camera Mouse uses correlation to track user-defined and automatically updating templates to track a small region in a video sequence.[7]

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.[8] The Vocal Joystick uses acoustic-phonetic parameters which are more appropriate for continuous input to control a cursor.[9] Inhyuk et al used image observation and EMG signals from probes attached to the user's neck.[10] Eagle eyes uses electrodes placed on the face to detect eye movement.[11] Vazquez et al used a head-mounted 3-axes accelerometer is used to aid in an eye tracking user interface.[12]

Several eye tracking user interfaces have also been studied and developed over the years using bio signals [13], and light reflection systems [12], among others [14, 15].

Many feature tracking user interfaces primarily use image streams from a regular camera. Recent advances in sensor technology have added a new dimension, literally and figuratively, to vision-based
applications that are designed to be readily available to consumers. In particular, the launch of the Microsoft Kinect in November 2010 spurred the development of various commercial applications and research work. A Guinness world record for being the fastest selling consumer device [16], development libraries, the following of major companies, and a long list of various applications that can easily be searched online, all indicate the popularity of this technology. It is not that depth sensors have not been utilized for research and applications before [17], but that a depth sensing technology with a relatively high resolution (640x480) is now made more readily available to researchers and consumers at a very low price. Some relevant works include those which use its depth data to detect and track face pose [18, 19] or to create 3D face models. [20]

3. FACING THE ACCURACY PROBLEM
The primary problem of face tracking user interfaces, inherited from its natural user interaction parent, is accurate translation to computer input. The problem is intensified by a small window of facial movement and the need to control a cursor in higher resolution screens. That screens continue to increase in size and resolution and that desktop applications tend to be more effectively used in high resolution does not help.

To illustrate this problem better, we estimated the span of normal face movement (without moving the torso and still is in convenient view of the screen) of a user about 24 inches from a screen and a 640x480 imaging sensor. Normal yaw and pitch movements of the face only spanned about 100x80 pixels. If face pose is needed yaw rotation for this user only spanned about 35°. So how can we make that small window operate on a 1024x768 or even a 1920x1080 screen?

To organize this discussion on improving accuracy, this user interface is logically separated into its data flow components as defined in the following diagram.

<table>
<thead>
<tr>
<th>User Input</th>
<th>User actions to be treated as input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Technology</td>
<td>Hardware and supporting software to capture raw input data</td>
</tr>
<tr>
<td>Retrieval of Feature Characteristics</td>
<td>Algorithms that perform and enhance feature detection and tracking</td>
</tr>
<tr>
<td>Processing of Feature Data</td>
<td>Algorithms that translate feature data into relevant computer input</td>
</tr>
<tr>
<td>Computer Input Behavior</td>
<td>How the computer should respond to processed input data</td>
</tr>
</tbody>
</table>

**Figure 1. Input components by data flow**

It should be noted that we acknowledge the presence of several other issues and considerations in this user interface. In particular, the question of usability would undoubtedly be a primary factor in the design decisions but it is intentionally not discussed here. The following discussion will be limited to how these different components affect accuracy.

3.1 User Input
A primary consideration for any user interface is what the user does that would be counted as an input. This affects the accuracy requirement on the feature processing component.

3.1.1 Cursor Control
For cursor control, a question is whether to use the feature’s pose (direction it is pointed towards) and/or its location in the projective plane that represents the camera’s viewpoint. It would be ideal to users and would provide greater flexibility if both feature location and face pose can be used as an absolute pointing mechanism that is robust to variance in the user's physical location in the workstation area. Using face pose would generally require a greater degree of accuracy primarily because it requires more than one data point or parameter to be accurate (e.g. two points that determine the line, or a plane and a point its normal intersects, etc.) whereas only one point is needed to be accurate for face location. While face pose could potentially make additional data available to process to mitigate the resolution problem, it would require that the face pose estimation be extremely accurate, which will be discussed later.

An issue on the use of face feature location in cursor control is that the normal movements of the face (without the torso moving) are generally comprised of rotations and not translations. So the location would have to be calculated from face yaw and pitch rotations. In addition, it is challenging to make the reference point dynamic enough. For example, if the reference point is, at least initially, the center of the sensor, a tracked feature that is not placed at this center may find that its neutral or resting position would cause the cursor to move when it should not. Camera Mouse solves this by having the user (or a user’s helper) specify the reference point by a mouse click. [7] However, the need for another individual to allow the use of the interface is not very convenient. Another solution is the implementation of a “physical interaction zone” suggested in [21] where the projected screen location is calculated based on the location of the tracked feature relative to other detected features. In a face tracking user interface that uses feature location, this means that the projective plane that is mapped to the screen could be limited to a small area on top of the user's body or shoulders. The reference point can also be adjusted using convenient calibration techniques (e.g. moving against the edges of the screen to move the reference point).

3.1.2 Selection Control and Other Commands
While cursor control by itself can provide selection based on dwell time, having other options available for selection would cater to the preference of a wider user base. Examples of facial gestures for selection or other special commands include opening the mouth, sticking out the tongue, raising eyebrows, and prolonged blinks, among others. As with other natural user interfaces, the Midas touch problem, where the user accidentally triggers commands from natural movement, has to be mitigated sufficiently. This requires accurate retrieval and processing of feature characteristics (see Section 3.3). It would also be helpful to program a gesture to turn the cursor or selection control on and off similar to the Snap Clutch of Istance et al.[22]

3.2 Input Technology
Various hardware can enable the input on a face tracking user interface. While there are potentially many such devices, we only
also able to keep track of objects in its practical field-of-view in
Segmentation of the foreground, also becomes a trivial task. It is
color/grayscale counterpart, it provides additional
Although the depth image from such sensors is more noisy pixel-
noise in its depth measurements make fine depth variation hard to
depth image have noise that make the edges unreliable for
in addition, surfaces that exhibit high specular reflection as well as external sources of IR light (e.g.
sunlight) will produce invalid depth readings.
which can break the characteristics of relevant objects due to
occlusion or camouflaging, among other things.

3.2.1 2D Image Cameras
A regular camera is utilized in many feature tracking applications. Much research has been done that deal with 2D images. 2D images from regular cameras have insignificant noise when it comes to object location. This means that, assuming a stationary camera, a stationary object and its boundaries will be in just about the exact same pixel location in the next captured image. In favor of this input technology is its ability for high resolution capture, not to mention its ubiquity. Noise is introduced by illumination changes and by the presence of other objects in the field-of-view that can break the characteristics of relevant objects due to occlusion or camouflaging, among other things.

3.2.2 Consumer Depth Sensors
It also has been noted that consumer depth sensors have recently gained popularity and outcomes that have been accomplished or are being worked on are done with the additional depth data available. Such sensors include the Microsoft Kinect and the Asus Xtion and are based on light coding technology developed by PrimeSense where depth calculations are done in parallel from reflection of structured infrared light.[23] While it is great for seeing small indoor scenes in 3D, noise in its depth measurements make fine depth variation hard to detect. The edges of objects in the depth image have noise that make the 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 invalid depth readings.

In the following sections, we particularly mention surveys of computer vision methods as they can be very helpful in understanding the level of accuracy these methods can come with. It should be kept in mind that this user interface operates in a constrained environment where the background does not really move, the imaging device is stationary, the user is the primary foreground object, and we are only tracking one blob. Occlusions other than the possibility of glasses and rotation will be uncommon and unexpected. Pre-processing methods like background subtraction by image differencing or depth segmentation are definitely helpful.

3.3 Retrieval of Feature Characteristics
Feature characteristics can be collected using various detection and other vision processing algorithms. This is a widely researched topic in computer vision and in no way does this paper attempt to give a comprehensive discussion on it. It is mentioned here for completeness as well as to give a high-level overview of noise considerations in the selection and use of such algorithms. We feel that this component is of utmost importance and requires a high degree of robustness. Anything short of that will easily make a poor user interface. As an interface inherently designed to be naturally learned, it is easy to learn that it is a poor one if it is.

One critical question is that of determining which feature to use for the interface. The answer depends on the input technology used as discussed in the previous section and how that feature can be robustly detected and tracked. For example, 2D images provide characteristically high intensity gradients for the eyes. In a noisy depth image, the eye on its own is harder to differentiate from another relatively flat patch. The feature or a combination of features have to be unique in the feature space. Using a combination of features can provide a confidence level that can be used to handle clutter that can cause algorithm failures.

In addition, algorithms used should ideally be able to process at the sub-pixel level in general. This mitigates the limited-sensor-resolution problem. While the ability to detect per frame shows the computational efficiency of an algorithm, we believe that this, in general, introduces noise and ignores valuable a priori knowledge. We believe it is ideal to have more robust detection which can then be tracked and which can be reconfirmed every now and then even at a cost of slightly, but only slightly, longer computation time.

3.3.1 Feature Detection
Zhang and Zhang published a survey on face detection in 2010 [24] to update a similar survey by Ming-Hsuan et al.[25] The survey on object tracking by Yilmaz et al also contains a general survey on object detection which can help in the selection of which feature to detect.[26]

The object detection framework of Viola and Jones [27] was a landmark work that made face detection more practical in real-time applications for its speed and effectiveness.[24] While much has been done since to improve face detection as explained in the survey cited, and since this is not meant to be a review of all possible methods, we evaluate only the suitability of the Viola-Jones detector in a face tracking user interface as an example. Note that other face detectors we have seen exhibit some of the same characteristics which are good to consider.

It is able to detect faces in real-time and on a per-frame basis with the use of highly efficient integral images and cascade classifiers. Particularly when the detection per frame is counted, it yields relatively high occurrences of false positives and negatives. Variability of the boundary of the detection frame also introduces noise in representing the face location. In addition, it is not robust to face rotation which is inevitable in a face tracking UI. All of these make it insufficient on its own for this UI.

Depth data can help simplify the detection problem by providing additional recognizable data. For example, false positives will be reduced if the depth data can confirm that the detected feature is not on a head. A 3D head can also be detected first then image processing can help detect and fix the location of additional features. In other words, for robustness, we recommend the use of multiple detection methods where one makes up for the weaknesses of the others. These methods should also be robust to at least a small degree of rotation invariance.

While the ability to detect per frame shows the computational efficiency of an algorithm, we believe that this, in general, introduces noise and ignores valuable a priori knowledge. We believe it is ideal to have more robust detection which can then be tracked and which can be reconfirmed every now and then even at a cost of slightly, but only slightly, longer computation time.

3.3.2 Tracking
For most users, face movement will be generally smooth and tracking can help realize this. Yilmaz et al did a survey on tracking methods in 2006.[26] It divided object tracking into point, kernel and silhouette tracking. The point tracking methods
defined there would only be appropriate for tracking smaller objects (e.g., eyes in a color image, nose tip in a depth image, etc.). Keeping in mind that feature shapes are generally the same throughout the sequence, kernel tracking methods would be more appropriate. Silhouette tracking can be useful for prompt face gesture detection (i.e., for selection control) and especially when feature detection is not done per frame. If depth images are used, remember that consumer depth sensors give noisy edge boundaries so those boundaries should not be used particularly when accurate estimation is required.

In addition, recent works with Iterative Closest Point (ICP) for tracking and modeling from 3D point clouds [18, 20, 28] shows potential in its ability to track fine movements of the face and even to create a personalized face template online which could make tracking more accurate.

3.3.3 Pose Detection
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. [29] Some of the fine pose estimation methods reviewed there looks promising. [30-32] However, as they noted, many of the works reviewed make assumptions or use methods that make them less applicable in real-world applications. These include limitation to a single rotational degree-of-freedom, requirement of manual intervention, familiar identity (i.e., where the test data is very similar to the training data), 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, we recommend the use of this survey in the selection of face pose estimation methods particularly because of the comparison on accuracy. The use of recent works on face pose estimation using depth information addresses some of these limitations and, with some improvement or modification, are appropriate in real-world face tracking user interfaces.[19, 20, 33]

3.4 Processing of Feature Data
Because the collected feature data is expected to have noise, it is essential to process it into data that is more appropriate as input. A cursor that jumps around and on and off the target will not be usable. A number of algorithms can be used to mitigate this noise on the calculated cursor point. The tracking problem in this case is much simpler than traditional object tracking in video sequences. Calculated cursor points come in sequence and the goal of the algorithm is to yield points that are smooth and are more representative of the user's actual face movement. While other methods could be used such as mean/median fillers, mean-shift, or particle filters, we believe that the Kalman filter is a very appropriate method to use in this scenario. The classic Kalman filter can operate on a simple time-discrete linear model such as this one and is relatively resilient to outliers and noise in general. 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 applicable in this case), new measurements or observations, and noise in the process and in the observations. A simple way of applying this filter to the stream of noisy cursor input data include 1) having the position and the velocity in both axes of the cursor define the state, 2) modeling the point location as a function of previous location and velocity, 3) using the new cursor points from the feature data as measurements, and 4) making sure that the noise distribution is not too far from the assumed Gaussian white noise. Similar techniques are helpful not only in the final calculated cursor point but in other data sets along the process flow as well (e.g., pose estimates, etc.)

3.5 Computer Input Behavior
Another question is whether to use absolute and/or relative cursor control. In the former, the feature data controls the absolute location of the cursor regardless of its previous location. In the latter, the data controls the location and possibly the velocity of the cursor relative to its previous location and velocity. Absolute cursor control requires more accuracy because the requirement scales with the screen resolution whereas relative control does not.

4. IMPLEMENTATION
Basic face tracking user interfaces were developed on top of existing face detection and tracking implementations. The goal of this is to understand the challenges on accuracy of the different UI components, thereby getting better insights to their solutions, and to validate the analysis done. Hence, we do not intend to implement the many options available with each component.

We are not so concerned about ground truth of the face location or pose as long as the cursor responds to general face movements and the user's eyes still get visual feedback of how the cursor is responding. For this paper, we are concerned with the ability to accurately target a specific region and so we analyzed the spread of the generated cursor points and how they are affected by various input components.

4.1 Experimental Setup
Both implementations were made and tested on a Windows 7 machine with an Intel® Core™ i5-2500 (3.3 GHz) CPU and 8GB of memory. Both were written in Visual C++. A standard 640x480 web camera was used and a Microsoft Kinect sensor.

For every combination of input options a subject was asked to look directly at a 23-in 1920x1080 screen and be still. Although, it is possible for noise to be present at different angles of the face, we exclude those options in this paper for brevity. We only used 5 seconds of data which was programmatically timed because we believe an attempt to aim a cursor should not last more than that. We retrieved the stream of calculated pair-wise points with whole values, which translates to screen pixels.

4.2 User Input
The cursor control was implemented using face location with a physical interaction zone defined by the physical limits of neck and face movements. This is called "Location" in the results. The second form uses the face pose data and implements an absolute pointing mechanism as described in Section 3.1.1.

4.3 Input Technology and Feature Retrieval
We used two existing face/head detection and tracking implementations - one from a regular camera and the other from pure depth generated by a Kinect sensor. It is not intended to compare the two implementations against each other as the comparison would not be on level ground. The purpose of using these implementations is to show how a face tracking UI can be
enabled by the two classes of input technology mentioned. They were chosen because they were found to be sufficiently robust to scale and rotation, they do face detection, tracking and pose estimation automatically, and they have available code or API that expose the face location and pose.

4.3.1 Using 2D Images

Seeing Machine's faceAPI [34] was used as the face tracking engine of this implementation. It is a commercial and proprietary product. As such, we cannot comment on how it accomplishes face tracking but that it does it very well. Among other things, it provides the location of the detected face and its orientation in radians, which is sufficient for the purpose of the user interface in study. True depth was provided manually as 24 inches where necessary as an estimate of the user's distance from the screen/sensor.

The location pointed to \((P)\) was calculated as follows:

\[
P_x = B_x \tan \theta_x + B_x
\]

Here, the pose vector is defined by a real-world point \(B\) and its Euler angle \(\theta\) (along the dimension defined by the subscript).

4.3.2 Using Depth Images

The face detection and pose estimation engine used for this implementation is the work done by Fanelli et al specifically with consumer depth sensors. [18, 33] They used forests of randomly trained decision trees [35] for both classification (head detection) and regression (pose estimation) using solely the depth data. It recalculates per frame and runs in real-time on a state-of-the-art desktop. The head pose in each frame of the sequence was determined using Iterative Closest Point (ICP) [36] with a personalized template constructed using an online model building approach by Weise et al.[28]

We used the default stride value of 5, which determines the balance between accuracy and speed. We discovered that speed is adversely affected by the inclusion of larger depth patches (e.g. user's body) and a shorter distance to the sensor, presumably because the training data had subjects at 1m away. In consequence, we had the subject sit from 1m away for a more real-time response. It was not expected that the pose estimation here will be accurate as published results with the best conditions still show a yaw error of 8.9±13.0°.[18, 33]

This setup gives real-world coordinates of the tip (\(T\)) and the base (\(B\)) of the pose vector. The location on the screen pointed to (\(P\)) can then be calculated as follows:

\[
P_x = \frac{T_x B_x - T_z B_z}{T_z - B_z}
\]

Here, \(P, T, B\) and \(z\) represent values along the dimensions \(x, y, z\), and \(z\) indicated by their subscripts where \(z\) represents the distance from the screen, \(x\) and \(y\) are on the axes that define the plane, and where the origin is at the center of the plane.

4.4 Processing of Feature Data

The Kalman filter implementation in OpenCV 2.3 was used according to the suggestions made in Section 3.4. The standard deviation of the calculated points without the Kalman filter was used to model the noise covariance matrices of the filter.

5. RESULTS

The following table summarizes the results obtained.

<table>
<thead>
<tr>
<th>Location Absolute Pointing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
</tr>
<tr>
<td>Depth</td>
</tr>
<tr>
<td>2D</td>
</tr>
</tbody>
</table>

The results show that the Kalman filter improved the accuracy of the cursor points in all cases. Using a single point face location was also more accurate except arguably with the depth method. This can be explained by the fact that the location was taken from the pose calculation which is the design and purpose of the algorithm. If location is used, it would have to be supplemented with an adjustment mechanism for the reference point as discussed in Section 3.1.1. The best combination (2D, location, w/ Kalman) allows the user to control the cursor well which is supported by its minimal variance.

It has to be noted that we separated the results for the horizontal and vertical axes to show that face tracking algorithms can respond differently on both axes and may have different noise models as we were able to confirm here with the 2D method. Understanding this difference helps in using more accurate noise models which we have done here with the Kalman filter.

6. CONCLUSION

We have enumerated the different input components of a face tracking user interface and discussed considerations for each on improving the accuracy of face tracking user interfaces. These components were implemented with some of the different options discussed. Similar to the suggestion given by Fanelli et al, [33] we confirm the benefit of using both image and depth data, especially when a device that can provide both is readily available, to make this user interface more robust. The implementations served to validate the understanding and the analysis of the challenges on making face tracking user interfaces more accurate, which would then serve as a stepping stone to usability studies. A statistical analysis on the accuracy of the said implementations were made that provided insight on some effective and useful methods that can be used in the design of such interfaces.

7. REFERENCES


