GlassGesture: Exploring Head Gesture Interface of Smart Glasses

Shanhe Yi, Zhengrui Qin, Ed Novak, Yafeng Yin†, Qun Li
College of William and Mary, Williamsburg, VA, USA
†State Key Laboratory for Novel Software Technology, Nanjing University, China
{syi,zhengrui,ejnovak,liqun}@cs.wm.edu, †yyf@dislab.nju.edu.cn

***Abstract*—We have seen an emerging trend towards wearables nowadays. In this paper, we focus on smart glasses, whose current interfaces are difficult to use, error-prone, and provide no or insecure user authentication. We thus present GlassGesture, a system that improves Google Glass through a gesture-based user interface, which provides efficient gesture recognition and robust authentication. First, our gesture recognition enables the use of simple head gestures as input. It is accurate in various wearer activities regardless of noise. Particularly, we improve the recognition efficiency significantly by employing a novel similarity search scheme. Second, our gesture-based authentication can identify owner through features extracted from head movements. We improve the authentication performance by proposing new features based on peak analyses, and employing an ensemble method. Last, we implement GlassGesture and present extensive evaluations. GlassGesture achieves a gesture recognition accuracy near 96%. For authentication, GlassGesture can accept authorized users in near 92% of trials, and reject attackers in near 99% of trials. We also show that in 100 trials imitators cannot successfully masquerade as the authorized user even once.**

I. Introduction

In recent years, we have seen an emerging trend towards wearables, which are designed to improve the usability of computers worn on the human body, while being more aesthetically pleasing and fashionable at the same time. One category of wearable devices is smart glasses (eyewear), which are usually equipped with a heads-up, near-eye display and various sensors, mounted on a pair of glasses. Among many kinds of smart eyewear, Google Glass (Glass for short) is the most iconic product. However, since Glass is a new type of wearable device, *the user interface is less than ideal*.

On one hand, there is no virtual or physical keyboard attached to Glass. Currently, the most prominent input method for Glass has two parts. However, each of these input methods suffers in many scenarios. First, there is a *touchpad* mounted on the right-hand side of the device. Tapping and swiping on the touchpad is error-prone for users: 1) The user needs to raise their hands and fingers to the side of their forehead to locate the touchpad and perform actions, which can be difficult or dangerous when the user is walking or driving. 2) Since the touchpad is very narrow and slim, some gestures, such as slide up/down, or tap can be easily confused. 3) When the user puts Glass on their head, or takes it off, it is very easy to accidentally touch the touchpad, causing erroneous input. Second, Glass supports *voice commands* and *speech recognition*. A significant drawback is that voice input cannot be applied in every scenario; for example, when the user is talking directly with someone, or in a conference or meeting. An even worse example is that other people can accidentally activate Glass using voice commands, as long as the command is loud enough to be picked by Glass. Additionally, disabled users are at a severe disadvantage using Glass if they cannot speak, or have lost control of their arms or fine motor skills.

On the other hand, *authentication* on Glass is very cumbersome and is based solely on the touchpad [1]. As a wearable device, Glass contains rich private information including point-of-view (POV) photo/video recording, deep integration of social/communication apps, and personal accounts of all kinds. There will be a severe information leak if Glass is accessed by some malicious users. Thus, any user interface for Glass needs to provide schemes to reject unauthorized access. However, the current authentication on Glass is far from mature: a "password" is set by performing four consecutive swiping or tapping actions on the touchpad similar to a traditional four digit PIN code. This system has many problems. First, the entropy is low, as only five touchpad gestures (tap, swipe forward with one or two fingers, or swipe backward with one or two fingers) are available, which form a limited set of permutations. Second, these gestures are difficult to perform correctly on the narrow touchpad, especially when the user is not still. Third, this sort of password is hard to remember because it is unorthodox. Finally, this system is very susceptible to shoulder surfing attacks. Any attacker can easily observe the pattern from possibly several meters away, with no special equipment.

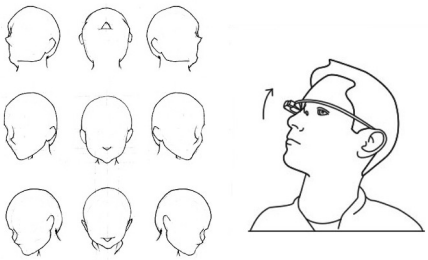

Fig. 1: Head Movements

To solve all of these problems, we propose the use of head gestures (gesture for short) as an alternative user interface for smart eyewear devices like Google Glass. Because head gestures are an intuitive option, we can leverage them as

978-1-4673-9953-1/16/$31.00 ©2016 IEEE

a hands-free and easy-to-use interface. A head gesture is a short burst of several discrete and consecutive movements of the user's head, as illustrated in Fig. 1. Motion sensors (i.e. the accelerometer and gyroscope) on Glass are able to measure and detect all kinds of head movements due to their high electromechanical sensitivity. However, smart eyewear presents new challenges for head gesture interface design. We need to answer questions such as "What are easy-to-perform head gestures?", "How do we accurately recognize those gestures?", "How do we make the system efficient on resource-limited hardware?", and "How does the system reject unauthorized access?" and so on.

In this paper, we present GlassGesture, a system aiming to improve the usability of Glass by providing a novel user interface based on head gestures. We are the first work, to the authors' knowledge, to consider head-gesture-based recognition/authentication problems for smart glasses. First, GlassGesture provides head gesture recognition as a form of user interface. This has several advantages against the current input methods, because head gestures are easy-to-perform, intuitive, hands-free, user-defined, and accessible for the disabled. In some situations, it may be considered inappropriate or even rude to operate Glass through the provided touchpad or voice commands. Head gestures in comparison, can be tiny and not easily noticeable to mitigate the social awkwardness. Second, the head gesture user interface can authenticate users. In particular, head gestures have not been exploited in authentication yet in the literature. We propose a novel head-gesture-based authentication scheme by using simple head gestures to answer security questions. For example, we ask user to answer a yes-or-no question, by shaking (no) or nodding (yes) her head. However, an attacker who knows the answer to the security questions can still access the device. To mitigate such attacks, we further propose to leverage unique signatures extracted from such head gestures to identify the owner of the device from other users. Compared to the original, touchpad-based authentication, our proposed head-gesture-based authentication is more resistant to shoulder surfing attacks , and requires much less effort from the user.

In summary, we make the following contributions:

- For gesture recognition, our system increases the input space of the Glass by enabling small, easy-to-perform head gestures. We propose a reference gesture library exclusively for head movements. We utilize activity context information to adaptively set thresholds for robust gesture detection. We use a weighted dynamic time warping (DTW) algorithm to match templates for better accuracy. We speed up the gesture matching with a novel scheme, which reduces the time cost by at least 55%.
- For authentication, we prove that "head gestures can be used as passwords". We design a two-factor authentication scheme, in which we ask users to perform head gestures to answer questions that show up in the near-eye display. To characterize head gestures, we identify a set of useful features and propose new features based on peak analyses. We also explore several optimizations such as one-class ensemble classifier, and one-class feature selection, to improve the authentication performance.
- We prototype our system on Google Glass. We design experiments to evaluate gesture recognition in different user activities. We collect a total of around 6000 gesture samples from 18 users to evaluate the authentication performance. Our evaluation shows that GlassGesture shows accurate gesture recognition. It can reliably accept the authorized users and reject attackers.

II. Related Work

Activity Recognition. Researchers have shown that when smart device is carried with user, it can provide context information about the user activities [2]–[4]. However, in this paper, we are not aiming at improving upon the state-of-the-art activity recognition systems. We use a simple activity detector, to tune parameters for gesture detection.

Gesture Recognition. It has been shown that gestures as input can be precise, and fast. While there is a broad range of gesture recognition techniques based on vision, wireless signal, touch screen [5]–[7], we focus mainly on motion-sensor-based gesture recognition because it is low-cost, computationally feasible, and easy to deploy on mobile devices [8]. We differ from these works in that we propose a head gesture based interface for smart glasses. And we carefully design the system to work with head gestures which faces different challenges such as noise from user activities, performance on resource-constrained devices. For head gesture recognition, existing work mainly focuses on vision-based methods [9], while GlassGesture utilizes sensor mounted on user's head. For gesture recognition on Google Glass, Head Wake Up and Head Nudge [10] are two built-in gesture detectors as experimental features which monitor the angle of head. A similar open-sourced implementation can be found in [11]. In contrast, GlassGesture is more advanced which can recognize self-defined, free-form head gestures efficiently and accurately.

User Authentication. There has been research on authenticating based on the unique patterns they exhibit while interacting with phone [12]–[17] through touch screens and motion sensors. These systems show that such authentication schemes are less susceptible to shoulder surfing, and, don't require the user to memorize passcode. For authentication on Google Glass, work [18] and [19] are touchpad-gesture-based authentication, which needs continuous user effort to hold up fingers on the touchpad. Our work is orthogonal that tries to bring easy authentication to smart glasses using head gestures, which is simple, hands-free, and requires less effort.

III. GlassGesture System Design

In this section, we present the system design of GlassGesture. First, we give an overview of our system and its architecture. Then we introduce each module and elaborate its corresponding components.

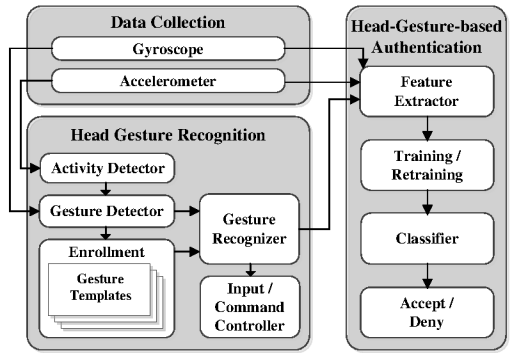

Fig. 2: System Architecture

A. System Overview

Our system consists of two modules, which together form our gesture-based interface. The first module allows users to input small gestures using their heads; the second module authenticates users based on their head gestures. The architecture of our system is illustrated in Fig. 2, which shows that the Gesture Recognition module is the corner stone. We leverage an activity detector to tune the parameters for more accurate gesture detection, based on user activity context. An enrollment submodule is in charge of managing the gesture templates. The gesture recognizer runs the DTW matching algorithm to recognize potential gestures. The gesture-based authentication module is built on top of the first module. It extracts features from the raw sensor data for training. With trained classifiers, we form a two-step authentication mechanism using simple head gestures to answer secure questions first and identifying the correct, unique signatures in the gesture movement data second. In the following sections, we present the design details of each module.

B. Head Gesture Recognition

Observations and Challenges. We have made some preliminary observations from the collected trace in Fig. 3: 1) Different activities add different amounts of noise. It is not easy to derive a general criterion for gesture detection in all of the many kinds of activities the user may be participating in at the time the gesture is made. 2) Head gestures mainly consist of rotations rather than accelerations. We see obvious gyroscope readings while the user is performing head gestures in various activities, compared to relatively noisy accelerometer readings. There it is possible to provide head gesture detection/recognition through the gyroscope data. 3) Head gestures can be used rather frequently by the user. We need an efficient recognition scheme for performance considerations.

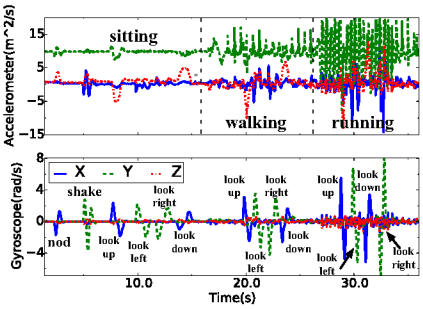

Fig. 3: Collected Sensor Trace: The user sits still for about 17s, then stands up and walks for about 10s, then runs for a few seconds and stops. In each activities (marked in accelerometer plot), she performs several head gestures such as nodding, shaking, looking up/down/left/right (sensor coordinate reference [20]).

In summary, we face three challenges in designing this module. 1) Head gesture library: There is no library, which defines the most suitable head gestures for smart glasses. 2) Noise: Sensors on Glass are used to collect head movements, while at the same time may also collecting noise from other user activities. This will deteriorate the performance of the gesture recognition. 3) Computation: In recognition tasks, computationally-intensive algorithms may need to be called frequently, resulting in unsatisfactory performance. Therefore, it must be optimized to be extremely efficient, without sacrificing substantial recognition accuracy.

Head Gesture Library. We need to provide a head gesture library as reference since head gestures are quite different from traditional hand gestures. For example, 1) head gestures mainly consist of rotational movement. 2) users moving their heads have limited freedom in 3D space. (e.g. usually humans can only look up and down in less than 180°. 3) In order to convey more information, we need a new set of head gestures beside the traditional ones that are already used (e.g., shaking for "no" and nodding for "yes"). In light of these constraints, we develop six basic candidate gesture categories adapted from work [8] and [21]: 1) nod, 2) look up/down/left/right, 3) shake, 4) circle, 5) triangle, and 6) rectangle. To clear up confusion when drawing (performing, acting out the gesture), we suggest the user move their head just like drawing something in the air in front of themselves using their nose like a pen tip.

Gesture	Styles	Number of strokes	Easy to perform	Frequency in Fig. 4	Easy to repeat	Decision
1	up and down	3+	5.2	low	no	keep
2	up/down/left/right	1	4.9	high	yes (81%)	keep,repeat
3	left and right	3+	4.4	low	no	keep
4	cw/ccw	1	3.0	very low	neutral	keep
5	cw/ccw, directions	3	2.2	very low	no	drop
6	cw/ccw, start points	4	1.4	very low	no	drop

TABLE I: Head gesture candidates.

With the purpose of trying to figure out what gestures are suitable, we performed a simple survey to rank them on how easy each category is to be performed for untrained users. It is important to note that the survey, and all data collections in the entire paper, have gone through the IRB approval process. In total, we have received 22 effective responses. The study results are presented in Table I. Our survey results indicate that nodding and shaking are popular and usually convey special meanings (e.g. "yes" and "no"). Circles are easy to perform since they are single-stroke. The rectangle and triangle gestures are the least favored, due to the multiple strokes they entail. Simple "look up/down/left/right" gestures are easy and fast, but they appear frequently in daily head movement as shown in Fig. 4, another study we have done to understand

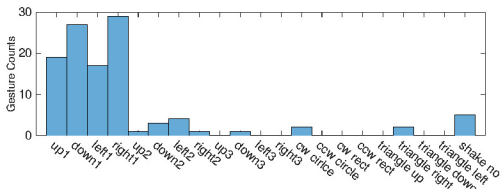

Fig. 4: Gesture frequency of a user seated, working at a desk for about 20 minutes. The number in the name is the repetition count. "cw" is short for clockwise, "ccw" is short for counter-clockwise.

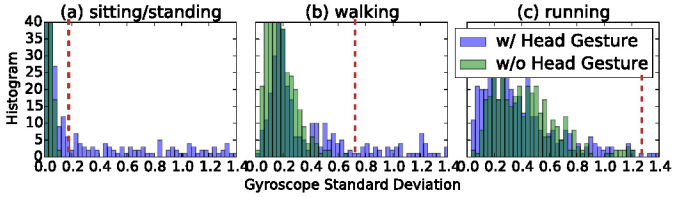

Fig. 5: Thresholds under different activities. The threshold will be set small when the user is sitting or standing, to enable even tiny head gesture detection (0.15). It will be set much larger when user is walking or running (0.7 and 1.3 respectively).

the frequency of daily life head gestures. This leads us to believe there will be a significant false positive rate if they are utilized naively. However, 81% of participants think they are easy to be performed repeatedly. We decide to keep these gestures, as long as the user is willing to repeat them two or three times consecutively to reduce the false positive rate. It is important to note that this head gesture library is only a default reference. GlassGesture allows the user to define new arbitrary head gestures. We also evaluate are gesture recognition system with "number" and "letter" input later in this work.

Activity Detector. The observations made in Fig. 3 motivate the need for a user activity context service, to help detect head gestures in different activity contexts. Normally, Google Play Service provides activity APIs, which can be leveraged. Unfortunately, it is not supported on Glass at the time of writing. To fill this gap, we have implemented a simple activity detector using the accelerometer. Samples from the accelerometer are chunked by an overlapping sliding window. We extract features, like *mean, standard deviation (std), root mean square (rms)*, from each axis in every chunk. Then a decision tree is used as the classifier, due to its simplicity and efficiency. The classifier currently gives one of the four outputs: 1) sitting/standing, which indicates that the user's head is fixed and the user's body is not moving; 2) walking; 3) running; and 4) driving. By using a 50 Hz sampling rate, and a 10-second window with a 5-second overlap, the classifier gives an average accuracy of **98%** in our preliminary experiments, which is adequate for use in our system.

Gesture Detector. The goal of the gesture detector is to capture potential gesture from the sensor's time series data. To find a potential gesture, we begin with windowing (30 samples) the gyroscope samples, and we calculate the rolling *standard deviation (std)*. A threshold on the gyroscope rolling std for the gesture detector will be set according to the current activity context, given by activity detector. To determine the thresholds, we collect user gyroscope data in different activities with and without the head gestures and apply a histogram-based method as shown in Fig. 5. In our current implementation, we disable the gesture recognition function when the user is running or driving for safety concerns. If the rolling std is below the current threshold, we know that there cannot be any gesture present, and the samples are discarded. Otherwise, we start to buffer both accelerometer and gyroscope readings. We keep these buffered samples until the rolling std drops below the threshold, indicating that user is no longer moving and the gesture has finished. We then check the sample length and drop all the buffered samples if the length is too short or too long (a head gesture usually ranges from 30 to 240 samples at 50 Hz sampling rate).

Gesture Recognizer. The **gesture recognizer** is the core of the gesture recognition module. A head gesture is defined as a time series of gyroscope samples about 0.5s to 2s long. The raw gyroscope sensor data, S, can be written as an infinite stream of four-tuples, i.e. $S = \{(x, y, z, t)_1, (x, y, z, t)_2, ...\}$. Likewise, a gesture G, is defined as a subset of sequential elements in S, i.e. $G = S_{t\in[t_1,t_2]} \subseteq S$. We refer to gestures that the system has already learned as "gesture templates", denoted as Gt. Because the system is passively listening, the user can perform any gesture at any time, so the problem becomes finding the gesture G in the infinite time series S and identifying which template Gt is the closest match.

1) Gesture Template Enrollment: GlassGesture selects templates, from the gestures recorded as the user does, in a gesture template enrollment procedure. This allows the system to be maximally accurate for its user. During enrollment, we require users to sit still when they are recording a new gesture. The recorded time series are normalized and error cases are filtered out. We will create templates from the recorded gestures using a clustering algorithm called affinity propagation, which has been proposed as an effective method [22]. The selected gesture, i.e. the affinity propagation cluster center, is stored as a gesture template in the system for recognition later.

2) Weighted Dynamic Time Warping: We use the weighted DTW algorithm to measure how well two gestures match, which has several advantages such as simplicity, working directly on raw data, and computational feasibility on wearables [8]. DTW calculates the distance between two gestures, by scaling one in the time domain until the distance is minimized. The algorithm takes two time series; a potential gesture G and a gesture template Gt. Assuming that G is of length l and Gt is of length l_t, where $i \in [1, l], j \in [1, l_t]$, given a 3-axis gyroscope time series, we have

$$\text{dtw}(G, Gt) = \sqrt{w_x D^2_{l,l_t}(x) + w_y D^2_{l,l_t}(y) + w_z D^2_{l,l_t}(z)}$$

The function D denotes the matching distance or cost, which is calculated as

$$D_{i,j} = d(G(i), Gt(j)) + \min\{D_{i-1,j-1}, D_{i,j-1}, D_{i-1,j}\}$$

where d is a distance measure; we use *Euclidean distance (ED)*. We also add weights (w_x, w_y, w_z) to each axis to better capture the differences of gestures, since we have found that

head gestures have different movement distributions along each axis. For example, a nodding gesture is much stronger in the x-axis the the y-axis or z-axis. Weights are calculated by the *std* on each axis of the template as

$$w_x = \frac{\text{std}(Gt_x)}{\text{std}(Gt_x) + \text{std}(Gt_y) + \text{std}(Gt_z)}$$

The best match (minimal $D(l, l_t)$) is optimized in the sense of an optimal alignment of those samples. We can say that G matches Gt, if $\text{dtw}(G, Gt)$ is below a certain threshold. To recognize which gesture is present in a given window, we need to run DTW iterating all templates. Whichever template has the lowest DTW distance with the target, and is below a safety threshold, is selected as the recognition result.

3) Efficient Similarity Search: DTW is a pair-wise template matching algorithm, which means that to detect a gesture naively, we need to traverse all gesture templates. It costs $O(N^2)$ to compare two time series at length of N (we set $l = l_t = N$ for simplicity), which is not efficient when there is a large number of gesture templates. We propose several schemes to optimize the performance.

Firstly, to reduce the search complexity, we want to build a k-dimensional (k-d) tree to do k-Nearest Neighbor (kNN) searches. However, tree branch pruning based on the triangle inequality will introduce errors if applied directly on DTW distances between gesture templates, since DTW distance is a non-metric and does not satisfy the triangle inequality [23]. Therefore, we build the tree using *Euclidean distance (ED)* instead, which is a metric distance, and therefore preserves the triangle inequality, allowing us to do pruning safely.

Secondly, to further reduce the computation, we down-sample the inputs before calculating the ED. Then we build the k-d tree. To recognize a target gesture, we first use the down-sampled target gesture to do the kNN search over the k-d tree. Then, we iterate over all k candidate templates to calculate the DTW distance with the target to find the best match with no down-sampling for the best accuracy.

The construction of a k-d tree is given in Alg. 1. And the kNN search is given in Alg. 2. Say we have m templates, which are all of length N. It costs $O(m * N^2)$ when iterating over all the templates to match a target gesture, using DTW. The set of m gesture templates in N-space (each template is of length N) can be firstly down-sampled to n_{ED}-space (each template is at n_{ED} length, $n_{ED} \ll N$). We build a k-d tree of $O(m)$ size in $O(m \log m)$ time to process the down-sampled templates, of which the cost can be amortized. The kNN search query can be answered in $O(m^{\frac{1}{n_{ED}}} + k)$, where k is the number of query results. In total, the time cost is $O(m^{\frac{1}{n_{ED}}} + k + k * N^2)$.

Lastly, we can also down-sample the gesture data before running DTW after the kNN search. The time cost will become $O(m^{\frac{1}{n_{ED}}} + k + k * {n_{DTW}}^2)$ where $n_{DTW} \ll N$ is the down-sampled length for DTW. However, it is non-trivial to choose proper n_{DTW}, since we don't want the down-sampling to remove important features of the time series. If this is the case, then the DTW algorithm may fail at differentiating two slightly different gestures. We evaluate n_{DTW} through our experiments in the evaluation section.

C. Head-Gesture-based Authentication

Basic Idea. As we mentioned previously, Glass does not have a robust authentication scheme. To secure the interface in GlassGesture, we propose the use of signatures extracted from simple head gestures. In order to lead the user to perform a natural and instinctual gesture, a "yes or no" security question, that can be answered using head gestures, is presented on the near-eye display. The user answers with head movements. In this way, the instinctual gestures (nodding and shaking) can be considered consistent head movements. After that, the answer (gestures) will be verified by the system. Features are extracted from motion sensors, then fed into a trained classifier. If the answer is correct and the classifier labels the gesture as belonging to the user, the user will be accepted. Otherwise, it will reject the user. Thus, we form a two-factor authentication scheme. While we mainly test the feasibility of the "nod" and "shake" gestures, since they convey social meanings in answering questions, we do not rule out the possibility of other head gestures. This scheme has several advantages over the existing authentication done on the touchpad. First, the user does not have to remember anything, as the signatures we extract are inherent in their movement/gesture. Second, nod and shake are simple gestures taking almost no effort from user. Finally, an attacker cannot brute-force this system even with significant effort, because 1) the near-eye display is a private display, which can prevent shoulder surfing on the secure questions; 2) the signature of the head gestures are hard to observe by the human eye, unaided by any special equipment. Furthermore they are difficult to forge even with explicit knowledge of the features.

Threat Model. We have identified three types of possible attackers. The *Type-I attacker* has no prior information whatsoever. This attacker simply has physical access to the user's Glass and attempts to authenticate as the user. Type-I attacks are very likely to fail and ultimately amount to a brute force attack, which can be mitigated by locking the device after a few consecutive authentication failures. The *Type-II attacker* may know the answer to the user specific security questions, but will try to authenticate with head gestures in their own natural styles (not trying to imitate the correct user's motions or features). The *Type-III attacker*, the most powerful attacker, not only knows the answers to the security questions, but also is able to observe authentication instances (e.g. through a video clip). The attacker can try to perform the gesture in a similar manner as the owner, in an attempt to fool the system. Note that, there is no security mechanism which can guarantee that the attacker will not be able to obtain the data on the device once the attacker has physical access. The proposed authentication method can slow the attacker down, foil naive or inexperienced attackers, and make the task of extracting data from the device more difficult.

Authentication Setup. In this offline setup phase, the user first needs to establish a large set of security questions with

Algorithm 1 Build KD-Tree

procedure BUILD KDTREES($\mathbf{T}$, n_{ED})
  for each template t in $\mathbf{T}$ **do**
    downsampling to length-n_{ED}
    stored in $\mathbf{T}_{down}$.
  end for
  Build a KD Tree from $\mathbf{T}_{down}$ using Euclidean distance, as $\mathbf{Tr}$
end procedure

Algorithm 2 kNN search.

procedure KNN SEARCH($\mathbf{Tr}$, t, k)
  put k nearest neighbors of target t in tree $\mathbf{Tr}$ into $\mathbf{C}$.
  for each candidate in $\mathbf{C}$ **do**
    run DTW on target and candidate.
  end for
  return index of minimal DTW distances
end procedure

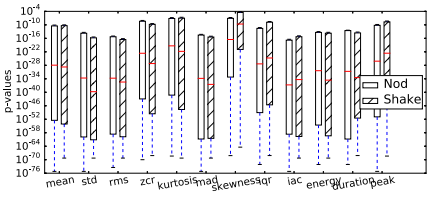

Fig. 6: K-S test results for gesture Nod and Shake

answers. These questions could be something like "Is Ford the maker of your first car?", "Is red your favorite color?" etc. Next, the user is also involved in contributing an initial training set, from which a classifier model can be built. Because the classifier requires some training samples before sufficient accuracy is achieved (>30 training samples in our evaluation), we optionally propose that the system can leverage the gesture recognition module to opportunistically collect instances of the "nod" and "shake" gestures. Whenever GlassGesture recognizes these gestures, we store these instances for classifier training in the authentication module.

Feature Set. We select statistical features such as, *mean*, *standard deviation (std)*, *root mean square (rms)*, *kurtosis*, *skewness*, *median absolute deviation (mad)*, *zero-crossing rate (zcr)* and *inter-quartile range (iqr)*. We also add features like *energy*, *duration* and *inter-axis correlation (iac)*. Additionally, we add a new category of features called *peak features* (including *average peak-to-peak duration*, *average peak-to-peak amplitude*, and *peak number*) by analysing peaks in the motion sensor data, which we have found effective at characterizing movements like head gestures. We collect motion sensor data of gesture samples from 18 users (gender: m/f: 14/4; age: 20-30: 11, 30-40: 5, 40+: 2.) while they are answering yes or no questions using head gestures. We extract features from the raw accelerometer and gyroscope data on each axis, in total 84 unique features, for each sample. To test the effectiveness of the selected features, we run a two-sample Kolmogorov-Smirnov test (K-S test) to see whether the features of different users are from differentiable distributions. From the results in Fig. 6, we can find that all the p-values, returned by K-S test, are smaller than the significant level (0.05), which indicates the effectiveness of selected features.

Training and Classification. SVM classifies have been widely used in biometric-based authentication systems and radial basis function (RBF) kernels have been shown to have good performance [13], [14]. For the authentication problem, a one-class classifier is the most practical model since, at the training phase, the system can only gather training samples from the authorized user. However, ideally, if the system is able to gather enough negative instances, the one-class classifier might be outperformed by a two-class classifier, eventually. Therefore, for practicality concerns, we report the one-class classifier results to assess our gesture-based authentication system. The training phase happens offline. We use a grid search to get the optimal parameters for the one-class SVM classifier (OCSVM) and the RBF kernel with a 10-fold cross validation. To further improve the classification performance, we employ a one-class ensemble SVM method (OCESVM) [24] to combine multiple classifiers. The basic idea is that we collect and rank multiple sets of parameters by the true positive rate (TPR) with a constraint on the false positive rate (FPR <1%) during the grid search. Then the top-r (we set $r = 5$ empirically) classifiers are chosen to form an ensemble classifier using majority voting on the classification decisions. We use the trained model to classify the test samples. The test samples can be labeled in one of two ways: 1) samples from the authorized user; 2) samples from some other, unauthorized user. We will present the evaluation of our training and classification in next section.

Feature Selection. While our features are extracted from three axes, it is possible that a gesture in 3D space may be well characterized by features extracted from data of only one (1D) or two axes (2D). Therefore, we apply recursive feature elimination (RFE) [25] to eliminate redundant or useless features, which will increase accuracy and reduce delay. In RFE, the training process will recursively select a subset of the features, which works best on preliminary data. However, RFE usually works with multi-class classifiers, not one-class classifiers. Therefore, we propose a new way of applying RFE in one-class classification. The training set in one-class classification are all positive instances (same class labels). The basic idea is to divide the training set into several groups evenly and manually assign each group a different virtual class label, to turn the one-class training set into a multi-class one. In applying of RFE onto this "fake" multi-class training set, we use a 10-fold cross validation and vote on the features in each run. Since features which top the voting result contribute most in differentiating those virtual groups, we eliminate features with more than e votes and finally train a one-class SVM classifier with the rest of features. The problem here is how to determine the value of e. Through our experiments we empirically set $e = 3$, which gives the best performance in most of our trials. We will evaluate this feature selection scheme later.

IV. EVALUATION

Currently, we have implemented GlassGesture as a Google Glass application. We adopt FastDTW implementation [26]

to build the gesture recognition module. For gesture authentication module, we compile libSVM [27] as native code to implement the classifier. The model is trained offline on a remote machine (MacBook Air, i5-1.3GHz and 4GB-RAM). In this section, we will evaluate our system in two modules individually. We will report performance metrics in terms of TPR ($\frac{tp}{tp+fn}$), FPR ($\frac{fp}{fp+tn}$), accuracy and F1-Score ($\frac{2tp}{2tp+fp+fn}$) for both gesture recognition and authentication.

A. Gesture Recognition

We prime our system with eight command gesture templates: *nod* and *shake*, *left* and *right* 3 times, *triangle* and *rectangle*, and *cw/ccw circle*. We also prepare our system for *number* and *alphabet* input. We choose those head gestures in order to evaluate the capability of gesture recognition on various gestures.

Gesture Recognition with command gestures. For each of these command gestures we conduct gesture data collection three times, once with as little head movement as possible (tiny) in sitting, and a second time with a normal/natural amount of head movement (normal) in sitting and a third time in a normal amount of head movement in walking. This experiment is repeated for multiple rounds with each round collecting about 10 gestures. The results of accuracy in Fig. 7 is in the form of confusion matrices.

1) Gesture in sitting: From results in Fig. 7 (a, b), we can see that for several gestures, such as *nod*, *left3*, *right3*, *shake*, the accuracy is perfect, even in the tiny gesture case. The reason behind is that the gesture has a repeating pattern in itself, which distinguishes it from other miscellaneous movements. The most easily confused gestures are *clockwise circle* and *triangle*, because of similar shapes in a clockwise direction. When the user tries to make her gesture very tiny, the head movement space is suppressed greatly, which will make these two gestures indistinguishable. Since our system allows users to define their own gestures, we can notify them in case new gestures are too similar to any pre-existing templates to ensure the best user experience.

2) Gesture in walking: When a user is walking, it is rather natural that the user will perform gestures in an obvious, unconstrained way. Otherwise, this gesture will just be buried in the noise of her walking. From the confusion matrix in Fig. 7 (c), we find minor deterioration of accuracy in recognizing gestures such as *right3* and *rect*, which we believe is caused by noise of walking movement. However, the *triangle* and *clockwise circle* are more distinguishable, which as we find is easier for the user to perform while walking rather than sitting.

Number and Alphabet Input. Next, we evaluate gesture recognition accuracy when we use head gesture as *number* and *alphabet* input method. Users are asked to draw *0-9* and *a-z* for at least 10 times to evaluate the accuracy. While **35** of total 36 gestures are 100% identified, the only error is one instance of number *9* is mis-recognized as number *7*. The failures are due to the limitation of template matching i.e. writing *7* and *9* are just too similar if user doesn't write them carefully using head

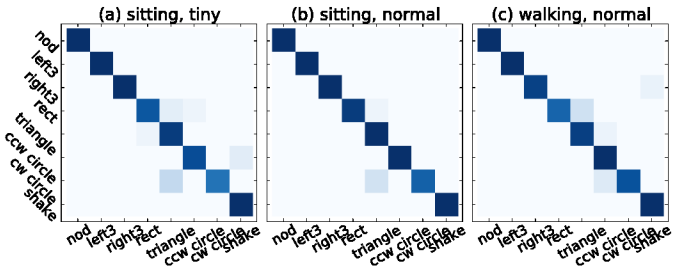

Fig. 7: (a) Confusion matrix of command gestures (sitting, tiny). TPR: 92.87%, FPR: 5.7%. (b) Confusion matrix of command gestures (sitting, normal). TPR: 96.99%, FPR: 2.4%. (c) Confusion matrix of command gestures (walking, normal). TPR: 94.88%, FPR: 4.6%

movement. One way to improve this is to explore different styles of writing them, which is out of the scope of this paper.

Gesture Recognition Performance. To demonstrate the performance of gesture recognition, we evaluate it with the processing of 36 gestures of *number (0-9)* and *alphabet (a-z)*. Firstly, we want to determine the proper down-sampling length n_{ED} for calculating *Euclidean distant* used in kNN search and n_{DTW} for calculating DTW used in template matching. In Fig. 8, we evaluate the gesture recognition accuracy at different down-sampling lengths (n_{ED}) and numbers of nearest neighbours (k). We found that when the n_{ED} is set as 10, it gives best accuracy. In Fig. 9, we change the n_{DTW} in the linear scanning using DTW distance metric. The time cost grows exponentially with the input length, while the accuracy can reach a satisfactory level when down-sampling length is as small as 40 or 50. Next, we show the processing speedup of our scheme against the linear scanning baseline. The results are shown in Fig. 10. We set $n_{ED} = 10$, $n_{DTW} = 50$. The number of nearest neighbours to be searched can be set to 10-14, which is a reasonable trade-off between processing speed and accuracy based on Fig. 10 and Fig. 8. The running time will be reduced by 70% when $k = 10$ and 55% when $k = 14$. We use $k = 14$ in our system.

B. Authentication Evaluation

We have collected motion sensor data from 18 users while they are answering questions using head gestures. We have gathered around 100-150 trials for each gesture of each user.Those data are pre-processed and used for feature extraction, model training and evaluation.

Impact of Number of Training Samples. Before training the model, we want to decide an appropriate size of training samples since it will be a trade-off between authentication accuracy and user convenience. We run the one-class SVM (OCSVM) training process with 10-fold cross validation. Based on trained models, we also build a one-class ensemble SVM classifier (OCESVM). As plotted in Fig. 12, we increase the percentage of training samples from 0.1 to 1.0, use the rest as test samples, and report average TPR and FPR of all users. We find that 30 samples (0.2 ratio) is sufficient to achieve an average TPR higher than 70% and keep an average FPR lower than 0.3%. OCESVM shows great gain of TPR and slight deterioration of FPR when the sizes of training samples

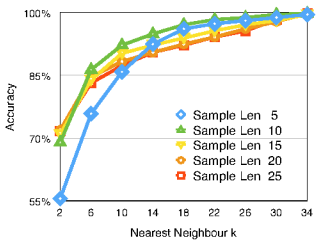

Fig. 8: Accuracy changes with sampling lengths (n_{ED}) and numbers of nearest neighbours.

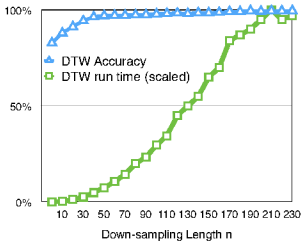

Fig. 9: Accuracy and scaled running time using DTW change with sampling lengths (n_{DTW}).

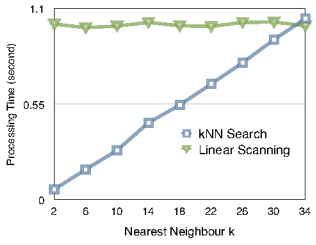

Fig. 10: Running time comparison between our scheme and linear scanning.

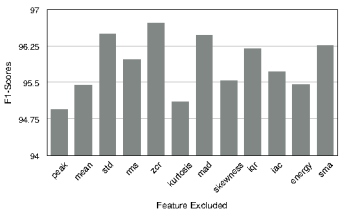

Fig. 11: The F1-score of certain category of features is excluded.

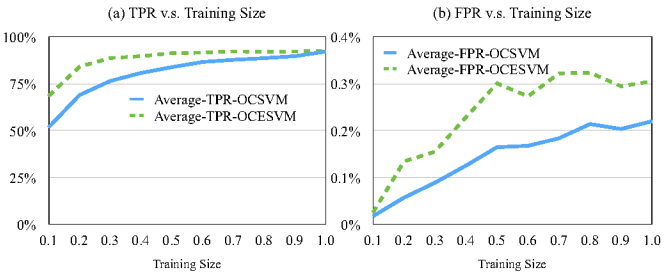

Fig. 12: The average TPR and FPR change with different ratios of training samples.

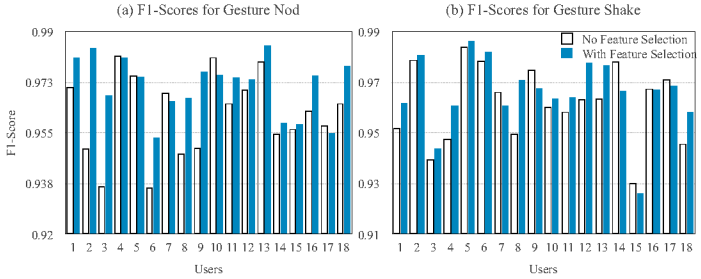

Fig. 13: F1-Scores of one-class SVM with or without feature selection for gesture nod and shake.

are small. Therefore, in our system, we build the training set passively and continuously in the background every time the user performs those gestures. We employ OCESVM when the size of training samples is insufficient, and fall back to OCSVM for system overhead concern when the gathered training samples are adequate. This scheme eases the burden of training on users significantly while maintaining high TPR and low FPR at the same time.

Single	TPR	FPR
GlassGesture Nod	92.43% (+/-3.09)	0.09% (+/-0.22)
GlassGesture Shake	92.33% (+/-3.32)	0.17% (+/-0.33)
GlassGesture Left3	89.08% (+/-6.36)	0.48% (+/-0.79)
GlassGesture Right3	89.61% (+/-5.99)	0.52% (+/-0.87)
Multiple and Comparison	**TPR**	**FPR**
GlassGesture (2 gestures)	99.16%	0.61%
Touchpad+ Voice (5 events) [19]	97.14%	1.27%
Touchscreen GEAT (3 gestures) [13]	98.2%	1.1%

TABLE II: FPR and TPR of authentication on two gestures.

Authentication against Type-II attacker. In order to understand the authentication performance, we evaluate the authentication against Type-II attackers, which are more powerful than Type-I attackers. We utilize the full size of the data set to train the model with 10-fold cross-validation for each user. While training model for a certain user, we use data samples from all other users as Type-II attacking trials. The result is shown in Table II in metrics of TPR and FPR and compared with several existing works. From the result, we can tell that our authentication system can identify authorized users with with a high TPR as average 92.38% and defend against Type-II attackers with a low FPR as average 0.13% if using Nod and Shake gestures. We are careful to bother no authorized user with occasional false positives. However, since gestures are very short, cost nothing, and are easy to perform, we assume that the user is willing to go through authentication multiple times which can basically eliminate the false positives. We compare authentication performance using two consecutive gestures with work [19] and [13], where both one class classifiers are used. Work [19] combines touchpad and voice commands to authenticate user in Google Glass. Our scheme requires fewer gestures, less effort, and produces better result. Work [13] is about touchscreen-based authentication on smartphone, while we show that we can achieve competitive performance using head gestures on Google Glass. Another work [18] uses a two-class SVM classifier, which only reports the average error rate (AER, defined as $\frac{1}{2}(1 - tpr + fpr)$) as 0.04 while using 5 touchpad gestures on Glass. Our scheme requires fewer gestures, and better result when multiple gestures are combined.

Impact of Peak Features. With the intention to investigate the impact of peak features, we use an feature-excluding method to verify the effectiveness of peak features. We collect number of true positive, false negative and false positive to calculate the F1-score as a metric to show the overall performance of the classification. In Fig. 11, the F1-score is lowered the most when peak features are excluded. Some other important features are *mean*, *energy*, *kurtosis* and *skewness*.

Impact of Feature Selection. To show how our feature selection method helps in our case, we compare F1-scores of classification with and without feature selection. From Fig. 13, we can find that for majority of users (13/18 and 12/18 respectively), feature selection improves the classification.

Imitator Attack (Type-III). In this evaluation, we want to

know whether an Type-III attacker, can fool the authentication system. We start by taking a short video of a victim while she is performing gesture-based authentication, and then present this video to attackers. We give attackers enough time to learn, practise, and mimic the victim. And we only start the authentication process whenever each attacker feels she is ready. We give 5 attackers 10 chances for each gesture and unlimited access to the reference video. In all of our tests (100 trials), attackers are never able to fool the system and (falsely) identified as authorized users. From the experiment, we find that an imitator fails in mimicking the head gesture because 1) it is not easy to recover every details of head gestures recorded by sensitive motion sensors through vision observation; 2) it is not easy to control the head movement precisely and make it like a natural movement during mimicking. The different muscle distributions of head and neck in human individuals will add different features to the sensor recordings.

C. System Performance

We report the running time of several important functions: DTW time cost in gesture recognition, training time cost (offline on a remote machine), and classification time cost in authentication. The time cost of gesture recognition grows linearly with the number of templates, while the time of running one instance of DTW is rather small as 30.2 ms. The training is offloaded to a remote machine and cost average 42.8 seconds per user, which is affordable since the request of training and retraining is relatively rare after the initial setup. Classification runs on the Glass, of which the cost (28.6 ms) of single instance is almost unnoticeable by users.

D. Other considerations

Due to space limit, we briefly discuss other practical considerations. 1) Authentication Frequency: The frequency is depend on the usage pattern of user. The default setting is to authenticate user after booting or being taken-off, which is a rather infrequent. 2) Biometric Invariance: We have been keeping collecting gesture samples from several users during a week. We have not noticed much difference in recognition/authentication accuracy. However, we do add template adaptation [8] and classifier retraining to our system in case of any performance deterioration. And we have fail-safe authentication for consecutive failures. We are still lack of enough data to claim that human head gesture is invariant in a long term. We leave those work in the future. 3) Power Consumption: Based on the energy consumption reported in [4] and [28], the battery life of constantly sampling sensors is 265 mins(300 mins daily in normal usage). We are expecting a much longer lifetime since our implementation is not always-on. The device will enter a low-power mode after a short period of inactive. It responses to wake-up events [10] and then the gesture interface will be enabled accordingly.

V. Conclusion

In this paper we propose GlassGesture to improve the usability of Google Glass. Currently, Glass relies on touch input and voice command and suffers from several drawbacks which limits its usability. GlassGesture provides a new gesture-based user interface with gesture recognition and authentication, which enable users to use head gesture as input and protect Glass from unauthorized attackers.

Acknowledgment

The authors would like to thank all the reviewers for their helpful comments. This project was supported in part by US National Science Foundation grant CNS-1320453.

References

[1] Google Glass Help, "Screen lock," https://goo.gl/Knf7pl.
[2] N. Ravi, N. Dandekar, P. Mysore, and M. L. Littman, "Activity recognition from accelerometer data," in *AAAI '05*, vol. 5.
[3] H. Lu, J. Yang *et al.*, "The jigsaw continuous sensing engine for mobile phone applications," in *Sensys '10*.
[4] S. Rallapalli, A. Ganesan, K. Chintalapudi *et al.*, "Enabling physical analytics in retail stores using smart glasses," in *MobiCom '14*.
[5] J. O. Wobbrock *et al.*, "Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes," in *UIST '07*.
[6] A. Colaço *et al.*, "Mime: Compact, low power 3d gesture sensing for interaction with head mounted displays," in *UIST '13*.
[7] Q. Pu, S. Gupta, S. Gollakota, and S. Patel, "Whole-home gesture recognition using wireless signals," in *MobiCom '13*.
[8] J. Liu, L. Zhong, J. Wickramasuriya, and V. Vasudevan, "uwave: Accelerometer-based personalized gesture recognition and its applications," *Pervasive and Mobile Computing*, vol. 5, 2009.
[9] L.-P. Morency and T. Darrell, "Head gesture recognition in intelligent interfaces: the role of context in improving recognition," in *IUI '06*.
[10] Google Glass Help, "Head wake up/nudge," https://goo.gl/6lfFWg.
[11] T. Horikawa, "Head gesture detector," https://goo.gl/uRgVnu.
[12] M. Frank, R. Biedert, E.-D. Ma, I. Martinovic, and D. Song, "Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication," *TIFS '13*, vol. 8.
[13] M. Shahzad, A. X. Liu, and A. Samuel, "Secure unlocking of mobile touch screen devices by simple gestures: you can see it but you can not do it," in *MobiCom '13*.
[14] C. Bo, L. Zhang, X.-Y. Li *et al.*, "Silentsense: silent user identification via touch and movement behavioral biometrics," in *MobiCom '13*.
[15] L. Li, X. Zhao, and G. Xue, "Unobservable re-authentication for smartphones." in *NDSS '13*.
[16] J. Sun, R. Zhang, J. Zhang, and Y. Zhang, "Touchin: Sightless two-factor authentication on multi-touch mobile devices," in *CNS '14*.
[17] Y. Chen, J. Sun *et al.*, "Your song your way: Rhythm-based two-factor authentication for multi-touch mobile devices," in *Infocom '15*.
[18] J. Chauhan *et al.*, "Gesture-based continuous authentication for wearable devices: the google glass case," *arXiv preprint*, 2014.
[19] G. Peng, D. T. Nguyen *et al.*, "Poster: A continuous and noninvasive user authentication system for google glass," in *Mobisys '15*.
[20] Google Glass, "Locations and sensors," https://goo.gl/Oj6Mqg.
[21] A. Akl, C. Feng, and S. Valaee, "A novel accelerometer-based gesture recognition system," *IEEE Transactions on Signal Processing*, 2011.
[22] A. Akl and S. Valaee, "Accelerometer-based gesture recognition via dynamic-time warping, affinity propagation, & compressive sensing," in *ICASSP '10*.
[23] B.-K. Yi, H. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," in *ICDE '98*.
[24] R. Perdisci, G. Gu *et al.*, "Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems," in *ICDM '06*.
[25] I. Guyon *et al.*, "Gene selection for cancer classification using support vector machines," *Machine learning*, 2002.
[26] S. Salvador and P. Chan, "Fastdtw: Toward accurate dynamic time warping in linear time and space," in *KDD '04*.
[27] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *TIST '11*.
[28] R. LiKamWa, Z. Wang, A. Carroll *et al.*, "Draining our glass: An energy and heat characterization of google glass," in *APSys '14*.