Challenges in Gesture Recognition for Authentication Systems

Gradeigh Clark and Janne Lindqvist
Rutgers University
gradeigh.clark@rutgers.edu and janne@winlab.rutgers.edu

ABSTRACT
In this paper we describe current popular gesture recognition methods that can be applied to gestures on mobile devices for authentication. Three different methods are considered : geometric recognizers, Hidden Markov Models, and Dynamic Time Warping. A brief description of each method is given along with a series of design considerations that we believe should be kept in mind when trying to develop recognizers for authentication.

Author Keywords
Gesture recognition; authentication systems

ACM Classification Keywords
H.5.2. Information Interfaces and Presentation (e.g. HCI): User Interfaces

INTRODUCTION
The proliferation of sensor-loaded interactive mobile devices like tablets, smartphones, and touchscreen laptops have allowed the generation of a variety of different gesture types that can be interpreted by these devices to perform any number of tasks. Broadly speaking, these tasks can be as diverse as tilting a screen for visual effect in a game or for changing screen resolution via pinching motions. There is prior work on defining gesture sets to accomplish some of these tasks that explore the applicability of both user-generated gesture sets and pre-defined gesture sets [5, 3, 9]. There do exist gesture sets have succeeded as standards from a 2D perspective: pinching for zoom, swiping to move a screen in a given direction, etc. In current smartphone designs, phones react to 3D gesture motions such as picking up the phone and moving it towards the ear to answer a call.

Gestures represent large potential as an authentication schema for smartphones, tablets, or any other sensor-rich device available in the marketplace. Gesture based methods have advantages over current popular authentication methods for devices (e.g. text entry or personal identification numbers) given that these gestures can potentially be done faster, require less concentration and lower accuracy while also being customizable, easier to remember and benefit from increased security. The most ubiquitous form of gesture authentication is the familiar nine pin password seen on Android devices.

Herein, we present on a number of popular recognition schemes that can be applied to mobile devices. We will narrow our focus to recognition techniques for 2D gestures. 2D gestures are gestures that use the touch screen for drawing a gesture (e.g. tracing a circle on the screen) and we will presume no restriction on the number of fingers that can interact with the screen. We examine three types: geometric methods [2, 1, 8, 6, 13], Dynamic Time Warping, and Hidden Markov Models [4, 7, 12]. These three appear to be the more prevalent types of recognizers used for the given gesture type, although that is not to say they are the only ones available for us; there are also Bayesian Networks, Artificial Neural Networks, Support Vector Machines, et cetera.

DESIGN CONSIDERATIONS
When considering gestures for authentication, we need to think about how a system should be configured to manage inputs. Drawing inspiration from Wobbrock [13] and others, we can make a list of design considerations for recognition with regards to authentication when designing a recognizer:

1. Sample invariant. Gestures that are recorded across different devices with their own configurations on how touch events are sampled along with the natural variances in the speed and time with which a user performs a gesture can lead to inputs that, can be spatially correct but mismatched with regards to vector size. The recognizer should resample the input such that it obtains an accurate portrait of the input data and normalize the size at the same time.
2. Trainable. A good recognizer should allow for the design and learning of the system to handle new inputs. It should not run only off a predefined set of gestures from the designer; users need to be allowed to design their own sets of gestures otherwise it is not possible to leverage the full utility of the password space afforded to gestural inputs on a mobile device.
3. Computationally feasible. When designing for mobile platforms, it is important to reduce the overall CPU and time consumption of the algorithm performing recognition. The user experience for authentication will be degraded if there is a delay for every sign in attempt, especially given how many times per day an average person now checks their smartphone.
4. Configurable. A gesture recognizer should allow for many different options for both the user and the developer. This could be small things like control over the sampling rate or how many stored templates used.
5. Resistant. A recognizer should be capable of of rejecting false users. To achieve this end, it may be more important

to leverage more biometric features of a gesture when performing the recognition. This could be anything from the finger length (as used to effect by Sae-Bae et al. [9]) to the pressure of the finger. Combining this feature with configurability would make a much more potent recognizer.

6. Ease of use. Although recognizers can be described in papers and pseudocoded, it may not be enough to communicate its effectiveness to developers. A difficult, mathematically intensive and hard to understand recognizer can cause issue with adoption especially so if the recognizer happens to be quite good. Most developers that may want to use it, say college students or startups, would have trouble picking the recognizer up if it is beyond their comprehension. So it is important to make a recognizer that is also clean, neat, and API friendly while trying – as much as possible – to reduce algorithmic complexity and bring the knowledge base a little closer to Earth.

There are also other considerations that need to be introduced due to the nature of smartphones and tablets and how users interact with them. These devices can be held in at least 4 different natural combinations (vertically up, vertically down, horizontally left, horizontally right), which can effect how gestures are interpreted if templates or features were extracted during training rounds were in a different orientation layout. These considerations are as follows:

1. Location invariance: No matter where the correct gesture is drawn on the screen, it should be authenticated correctly.
2. Scale invariance: No matter what size the correct gesture is drawn to on the screen, it should be authenticated correctly.
3. Rotation invariance: No matter what angle the correct gesture is drawn at on the screen, it should be authenticated correctly.

We would argue that location and scale invariance are important when dealing with cross-platform authentication, because the screen dimension inherently limits what size the gesture can be drawn to and the area over which a gesture can be performed would cause wild variations in where it would be drawn depending on the user. Rotation invariance is useful for reducing computational complexity when dealing with individualized free-form gestures as we had.

How do all of these design choices affect the security of the authentication procedure, in terms of being able to authenticate correctly and not letting false authentication attempts through, remains an open question. Therefore, one obvious way to move forward is to relax and restrict the above choices, and try out the performance with participants.

Speaking heuristically, however, there are obvious tradeoffs. Location invariance sounds quite strict; this requires the user to draw a gesture the same place every time. But this actually is not uncommon; the Android nine pin configuration system already employs this. But the problem there is the fact that the password space is more limited and there are cues for recall, lowering overall security. A more ideal case for a secure gesture input to the device would be a multitouch one where the user draws without the trace being revealed on the screen and without constraint on path as in the Android layout. In this instance, location invariance can be a hindrance; with a free-form gesture like the one described, a user needs to remember exactly where to draw on the screen or otherwise face repeating their trials. Some users could prefer this, hence why configurability should be an option. We do not envision this being as big an issue on a smartphone as it would be on a tablet since the screen size creates more possibilities.

As far as scale invariance, this is an issue that should be addressed in some cases. Algorithms that are scale invariant, while they pass the correct gestures through in spite of their size, also allow other anomalies to happen. Rectangles get interpreted as squares; circular shapes are all condensed to one size. It is also questionable whether or not users can comfortably remember the correct size (at least, after some trials) to draw the gesture to each time. There is some idea that can support this; after enough practice, a user should be able to get most things to the same size every time – same idea behind having consistent handwriting.

Rotation invariance, similar to scale invariance, introduces anomalies as well. An arrow that is drawn facing right would be interpreted the same as an arrow drawn in any other orientation. So any type of directional cues are lost in translation with regards to rotation invariance. Out of the three listed, this should be the easiest one to relax from a user's perspective since it seems intuitive that an 8 should fail to authenticate against an infinity symbol.

GESTURE RECOGNITION TECHNIQUES

Geometric Methods

The current popular methods for this space are the $1 - Recognizer [13] and its appropriate derivatives [2, 1, 6]. Important notice should be given to Rubine [8], who laid the ground work for $1, and is included in this section even though Rubine [8] extracts more features as well for recognition. These methods are effectively a nearest neighbor, template matching approach.

1. These template matching algorithms will begin by resampling the raw gesture to N points, where N is some integer.
2. Afterwards, the resampled gesture is rotated until the angle that the first point in the sequence makes with the centroid of the gesture sequence is zero degrees.
3. The gesture is then translated to the center of the 2D plane it occupies.
4. The gesture then has its size normalized so the points are contained within a unit box.

Users need to enter a number of templates that can be used to train the recognizer, each of which go through the above process. Once there are training templates in the system, inputted gestures undergo the next steps:

1. The recognizer will then do a distance comparison (Euclidean, cosine, Mahalanobis, et cetera) between successive points and aggregate them to compute a score [2],

2. However, after performing the score computation, the gesture needs to be continuously rotated by some angular distance and have another score computed for each rotation - this is to find the best score when comparing to the template, whose orientation may not be the same as the inputted gesture even after normalization [6],[13].

Protractor [6] gets around the last step by solving a minimum angular distance problem and computing the score after that - much quicker than what is seen from Rubine [8] or Wobbrock [13].

Hidden Markov Models

Markov Models contain a discrete series of states and state transitions that correspond to measurable events[4, 7]. For gesture recognition, these events would be processes related with drawing the gesture. For a normal Markov Model, there is a state transition given a known, measurable event – for example, when using a vending machine; unless the appropriate amount of measurable change is present when selecting an item there will be no transition to a state where someone receives what they want from the machine. Hidden Markov Models (HMMs) are called "hidden" because there is no way to measure how the gesture is transitioning based on the input data, hence we must make an estimate as to which state to move to.

Instead, there are multiple possible transitions. There is a probability matrix that describes the chances of transitioning to one state from a given state. A gesture class would be classified with a sequence of states (say, move up or move down, etc.) and probabilities corresponding to a series of state transitions are calculated and compared against each other. Where the gestures are defined and specified by predetermined HMMs. For the system to learn a HMM for a gesture, it needs to take gesture data from the user and adjust the model probabilities to get the best idea for the state transitions. The model that is trained correctly can be used to evaluate and classify the incoming gesture[4, 7]. Downsides to this implementation is that HMMs require a large number of training examples [12] and authenticate slower than the previous two methods [13].

Dynamic Time Warping

Gestures typically record time data along with coordinate points. The gestures tend to be mismatched because users do not enter their gestures at the same rate on successive tries - a plot of the x gesture versus time for two attempts would show that they do not line up and thus they cannot be evaluated against each other; one figure would be a time shifted version of the other, assuming correct inputs. Dynamic Time Warping(DTW) scales the gestures in time such that they can be compared correctly – a matrix is constructed to align the time series path that is filled with these distances or "costs". The matrix is populated with the time difference between successive points in each series – if two gestures have time length of N and M respectively, the matrix is of size NxM consisting of distances between all points in one series with respect to the other. [11, 9].

DTW then works to create the lowest cost path or "optimal warp" between points. This is done by traversing the matrix and imposing boundary conditions; it must start at some corner origin (i,j) of the matrix and end up at another corner (n,m). The algorithm is restricted to not jump in the time index and it cannot go backwards. Assurances are built into the computations such that the traversal path does not stray far from the diagonal of the two sequences in the matrix. The average variance and standard deviation for a gesture against the template is then computed and measured against a threshold to determine authentication [11, 9].

CONCLUSIONS

We have outlined three popular methods that broadly define the gesture recognition space for 2D gestures that can be done on smartphones or tablets along with design considerations for what features an ideal recognizer strives for.

Geometric methods are the simplest and easiest ones to understand; distance based measures that are computed as scores. Computationally, these are also feasible; the resampling can use as little as six points [6] without severe degradation to recognition capability, allowing for fast scoring. It is difficult to get beyond anything more with these, however; they do not require biometric input data and they only work using one set of features – the spatial component of the gesture. At most, current geometric recognizers allow optionalities for rotation invariance but not location or scale. Geometric recognizers currently see some industry use through being implemented in the Android library's classes on gestures.

HMMs are the most mathematically complex and difficult to understand of the ones presented in this paper. Preparing and defining states for the different gesture types as well as algorithmically setting up probability matrices requires training data from the user. They tend to scale well aftet training since new inputs do not affect prior ones; the vocabulary of the model only increases. Tradeoffs, though, are that they require a lot of data to maintain and are not effective until there are already a larger number of training templates than the other two methods; they are computationally weighty. Whitehead et al. [12] concluded that, for their HMM gesture recognizer, a developer should have 250 training samples optimally and around 27 transition states for their 3D recognizer. However, they represent the most possibility of the three presented here to achieve the range of design considerations since the model can use many features to make choices about the state transitions. HMM see adopted use (along with SVM, mentioned earlier) for use with gesture shortcuts (e.g. pinching to zoom and swiping to pan the screen) – these are easier since the set is predefined.

DTW is a hungrier method as compared to geometric recognition. In general, DTW algorithmic complexity obeys a square law, and the template storage is on par with HMM. For mobile devices, this is not recommended on this stance alone. As far as design characteristics, DTW examines differences usually in space or in time but does not give consideration to other features. DTW similarly requires the use of stored templates for matching, as in the geometric method, and can

Criteria	Geometric	HMM	DTW
Sample Invariant?	Yes	Yes	Yes
Trainability	Low	Low	High
Computation	Low	Medium	High
Configurable	Low	Low	High
Resistance	?	?	?
Ease of Use	High	Medium	Low

Table 1. Summary Table. Note that resistance is left as a '?'; this is a highly subjective category that is difficult to properly measure. It will require much future work in a controlled environment to properly determine.

require more templates than that method to achieve similar results [6, 13].

There is not a clear winner yet for a ubiquitous recognizer that can be used strictly for authentication given the considerations outlined. A free, open design space for gesture recognition techniques has given rise to the the novel innovations by researchers outlined in this paper and as a result methods in this area are generated on an ad hoc basis. Ad hoc is used here to mean that researchers decide they want to begin the implementation of a new type of recognizer by utilizing using different features rather than authentication versus whats currently published and show how well it works in comparison. On the flip side of that, entrants to the field who want to implement gesture recognizers have a wide selection and have to do additional work to decide which method they need to use for their application.

It can be suggested that HMM is the most optimal type of recognizer to be used for more varieties of applications but is greedy and more difficult to program. This can additionally be argued with DTW since it requires complicated processing steps on data (e.g. matrix operations).

As far as authentication is concerned, it is best for an entrant to the field to attempt these geometric algorithms since they are neat, ordered, and simple to program. Given the level of customization present in free-form gestures (as far as the appearance of the gesture is concerned), geometric recognizers will work fine when dealing across templates. Without prior knowledge of the gesture itself, an attacker to the device would be stopped by the recognizer unless they were capable of generating a gesture close enough to the stored templates and if that is the case then it's more of a question on the security content of that stored gesture (Is it too simplistic, e.g. a circle?) than a judgment on the recognizer. One way to quantify the security of a generated gesture would be a recent one presentd by Sherman et. al [10]. They generate a security score by measuring the mutual information between a gesture and its template after the removal of "predictable" features.

Overall, a more fluent design approach would be to try to hit goals outlined in the design considerations and outlining how a recognizer succeeds and how it fails when moving ahead to design ones for authentication purposes.

REFERENCES

1. L. Anthony and J. O. Wobbrock. A lightweight multistroke recognizer for user interface prototypes. In *Proc. of GI '10*.
2. L. Anthony and J. O. Wobbrock. $n-protractor: a fast and accurate multistroke recognizer. In *Proc. of GI '12*.
3. C. Bo, L. Zhang, X.-Y. Li, Q. Huang, and Y. Wang. Silentsense: Silent user identification via touch and movement behavioral biometrics. In *Proc. of MobiCom '13*.
4. J. Fierrez, J. Ortega-Garcia, D. Ramos, and J. Gonzalez-Rodriguez. HMM-based on-line signature verification: Feature extraction and signature modeling. *Pattern Recogn. Lett.*, dec 2007.
5. S. A. Grandhi, G. Joue, and I. Mittelberg. Understanding naturalness and intuitiveness in gesture production: insights for touchless gestural interfaces. In *Proc. of CHI '11*.
6. Y. Li. Protractor: a fast and accurate gesture recognizer. In *Proc. of CHI '10*.
7. D. Muramatsu and T. Matsumoto. An HMM on-line signature verifier incorporating signature trajectories. In *Proc. of ICDAR '03*.
8. D. Rubine. Specifying gestures by example. In *Proc. of SIGGRAPH '91*.
9. N. Sae-Bae, K. Ahmed, K. Isbister, and N. Memon. Biometric-rich gestures: a novel approach to authentication on multi-touch devices. In *Proc. of CHI '12*.
10. M. Sherman, G. Clark, Y. Yang, S. Sugrim, A. Modig, J. Lindqvist, A. Oulasvirta, and T. Roos. User-generated free-form gestures for authentication: Security and memorability. In *Proceeding of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, 2014.
11. J. Tian, C. Qu, W. Xu, and S. Wang. Kinwrite: Handwriting-based authentication using kinect. In *Proc. of NDSS '13*.
12. A. Whitehead and K. Fox. Device agnostic 3d gesture recognition using hidden markov models. In *Proceedings of the 2009 Conference on Future Play on @ GDC Canada*, Future Play '09, pages 29–30, New York, NY, USA, 2009. ACM.
13. J. O. Wobbrock, A. D. Wilson, and Y. Li. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proc. of UIST '07*.