Modern Approaches to Gesture Recognition

Scott Steffes
Department of Computer Science
University of Minnesota, Morris
600 E 4th St.
Morris, Minnesota
steff298@morris.umn.edu

ABSTRACT

Gesture recognition refers to the computerized processing of data from human actions to determine whether the data corresponds to a particular human-readable gesture. This data is provided via an input device such as a camera, accelerometer, or touch screen. To accomplish gesture recognition, a computer must classify input data as corresponding to either no gesture, or one of a list of discrete gestures determined beforehand. Gesture recognition seeks to provide an alternative or supplement to traditional human-computer interaction (HCI) methods. Effective gesture recognition systems can allow users to intuitively specify operations that are cumbersome to specify using a traditional mouse and keyboard. Gesture recognition systems can also make computer interaction possible for those who can not operate a mouse and keyboard. This paper gives a brief overview of the gesture recognition field and reviews three modern approaches to gesture recognition.

Categories and Subject Descriptors

H.5.2 [**User Interfaces**]: [Interaction styles]

General Terms

Human Factors

Keywords

gesture recognition, human-computer interaction, hidden markov models, dynamic time warping

1. INTRODUCTION

Through traditional methods of human-computer interaction (HCI), it is possible for the average user to specify the operations required to complete many useful tasks. Often, however, the mouse and keyboard fail to make use of the human body's incredible expressive potential. To use most computer programs, users must learn to express their intentions using computer-friendly keyboard shortcuts and menu structures. The field of gesture recognition seeks to improve various aspects of HCI by leveraging the powerful human ability to specify intent through gesture. For example, the task of text manipulation can be made more intuitive by allowing the user to specify actions using proofreader's marks using a pen/tablet interface [6]. Gesture recognition can even open up new avenues of HCI that didn't previously exist such as recognizing American Sign Language. For users with physical disabilities, interacting with a computer via keyboard and mouse may be extremely difficult. Gesture recognition can allow such users to operate a computer by decreasing the need for extremely precise movement.

Not surprisingly, producing a method of interaction capable of capturing the nuances of human gesture with a computer is fairly difficult. We must look beyond traditional input methods and employ technologies such as cameras, accelerometers, and touch screens. In addition, we must develop approaches to analyze the rich sense data coming from these technologies. In this paper we give a brief overview of the gesture recognition field and review three modern approaches to gesture recognition. These approaches were selected because they expand on commonly employed approaches, either by modifying the algorithms involved or applying existing algorithms to new problem spaces. We begin by describing concepts that are key to all forms of gesture recognition then go into greater detail on the three selected approaches.

2. BACKGROUND

To simplify the task of explaining gesture recognition approaches, we will begin by defining a set of terms commonly used in the gesture recognition field. A *gesture* is a single instance of captured human motion. Each gesture is typically represented as a discrete collection of sense data. A *gesture class* is a set of gestures that a user considers meaningfully similar and, as a whole, distinct from other sets of gestures. For example, "Sway arm horizontally from left to right" is a gesture class that various similar gestures could belong to. Though gesture class implementations vary greatly between approaches, they all have some notion of a typical gesture of that class against which new gestures can be compared. An *unclassified gesture* is a collection of sense data that corresponds to a single unknown gesture. The core function of a gesture recognition system is to take in unclassified gestures and classify them. A *pre-classified gesture* is a gesture that has been classified by a human for training purposes. A *gesture recognizer* is a function associated with a particular gesture class that takes in an unclassified gesture and returns a value indicating to what extent the unclassified gesture matches the gesture class.

While the approaches in this paper have differing input methods and classification algorithms, they share a common structure. Each approach begins with a method for producing and training gesture recognizers. Such a method takes in multiple pre-classified gestures and returns a set of gesture recognizers, each trained for a particular gesture class. At this point, the recognizers are ready to be applied to new

data. Given an unclassified gesture, the gesture recognizers are applied to the new gesture, each returning a value. In the best case, a single recognizer indicates a close match while the other recognizers indicate poor matches. When this occurs, the decision to classify is straightforward. In another case, no recognizers indicate a close match. At this point, the approaches differ slightly. In some approaches, the gesture will be placed in the closest gesture class, running the risk of a misclassification. In other approaches, the gesture will be rejected, running the risk of failing to classify a gesture that is, in fact, a member of one of the gesture classes. How an approach deals with this scenario is dependent on the problem space it is applied to. In cases where more than one recognizer indicates a close match, more complex sense data or a more powerful recognition algorithm may be necessary.

3. LINEAR EVALUATION FUNCTIONS

In the simplest implementations of the above approach, each gesture is represented as a vector of features where each feature is a single numeric value. Geurts *et al.* [3] describes an application of this approach involving the Nintendo Wiimote. The Wiimote is a handheld device that contains an accelerometer and a gyroscope. The Wiimote returns its rotation (yaw, roll, pitch) and velocity (x,y,z) data every 10ms. Through trial and error, the authors selected four features to represent each gesture. From a set of Wiimote measurements taken over time, a gesture was created by taking the following 4 values:

1. the sum of the yaw data points
2. the sum of the roll data points
3. the sum of the pitch data points
4. the sum of the accelerations along the z-axis

and storing the results in a vector. It is fairly simple to compare and average these feature vectors and, as we will soon see, this greatly simplifies the gesture recognition task.

For their gesture recognizers, the authors represented each gesture class as a single feature vector produced by averaging the feature vectors of pre-classified gestures in that class. For example, the yaw value of the gesture class is calculated by taking the arithmetic mean of the yaw values of all the pre-classified gestures in that class. To compute how well an unclassified gesture fits in a gesture class, the authors calculate the *Mahalanobis distance* between the unclassified gesture's feature vector and the gesture class feature vector. The Mahalanobis distance between two vectors is similar to Euclidean distance, but is scale-invariant, meaning features with greater variance do not disproportionately affect the distance measure. Mahalanobis distance also takes into account the correlations between features. A small Mahalanobis distance indicates a close match between a gesture and a gesture class. [6]

In many Wii games that encourage the user to exercise by making large gestures, it is possible to cheat and avoid exercise by making small gestures. This undermines these games' intent to promote healthy behavior. In Geurts et al. [3], they attempt to solve this problem by designing gesture recognizers that take into account not only the shape, but also the size of gestures. They want to force the user to make large gestures by not recognizing correctly-shaped small gestures. To this end, their choice of features was effective. In real-world testing their system was able to correctly identify 92% of users' gestures. After subjecting users to strenuous exercise, their ability to produce sufficiently large gestures decreased and the recognition rate dropped to 63% suggesting that the system was able to weed out small gestures. The gesture set they chose was arguably easy to recognize as it consisted of fairly distinct, simple gestures. The authors suggest that their system would not perform well with a larger, more complex gesture set. Because the features they choose are simply sums of data points, the features do not capture any temporal information about the gestures. For example, a gesture with multiple steps would have the same feature values regardless of the order in which the steps were performed. Using a similar approach, Rubine [6] was able to effectively recognize a larger set of pen/tablet gestures using 13 features consisting of more complex mathematical and trigonometric functions of the raw input data.

4. GESTURE RECOGNITION USING DYNAMIC TIME WARPING

4.1 Representing a Gesture as a Time Series

While representing a gesture as a vector of features is sufficient for some gesture recognition scenarios, it is sometimes helpful, and arguably more intuitive, to represent a gesture as a list of measurements taken over time. This type of representation is known as a *time series*. A common form of time series data seen in gesture recognition is the output from an accelerometer. At regular intervals, an accelerometer measures and reports its velocity in 1, 2, or 3 dimensions. Each measurement taken individually reveals little about the motion it came from, but the measurements organized chronologically as a time series produce a rich motion profile. [1]

Gesture recognition using time series is considerably more complicated than gesture recognition using feature vectors for a number of reasons. Producing a gesture class representation against which unclassified gestures can be compared is non-trivial. Unlike feature vectors, there is no standard way to find the average of a set of time series. For this reason, Fu *et al.* [2] calculate the distance between an unclassified gesture and a gesture class by first calculating the distances between the unclassified gesture time series and each of the pre-classified gesture time series for that class and then taking the arithmetic mean of those pairwise distances. While this is computationally costly, it simplifies the problem to calculating the distance between pairs of time series.

To illustrate common problems encountered when calculating the distances between time series, we will start with a naive approach, enumerate the problems with that approach and propose solutions to those problems.

Given two time series (A, B), a naive approach to calculating the distance D between them is:

$$D = \sum_{i=1}^{n} |Dist(A_i, B_i)|$$

where n is the length of the longest series, A_i is the ith element of A, B_i is the ith element of B, and $Dist()$ calculates the distance between two elements. This method has several problems. First, if A and B are not of the same length it

Table 1: An example warping matrix and warping path aligning the time series $A = [1, 2, 2, 4, 5]$ and $B = [1, 1, 2, 3, 5, 6]$. The warping path is highlighted. Adapted From [2].

1	**0**	**0**	1	5	21	46
2	1	1	**0**	1	10	26
2	2	2	**0**	1	10	26
4	11	11	4	**1**	2	6
5	27	27	13	5	**1**	**2**
	1	1	2	3	5	6

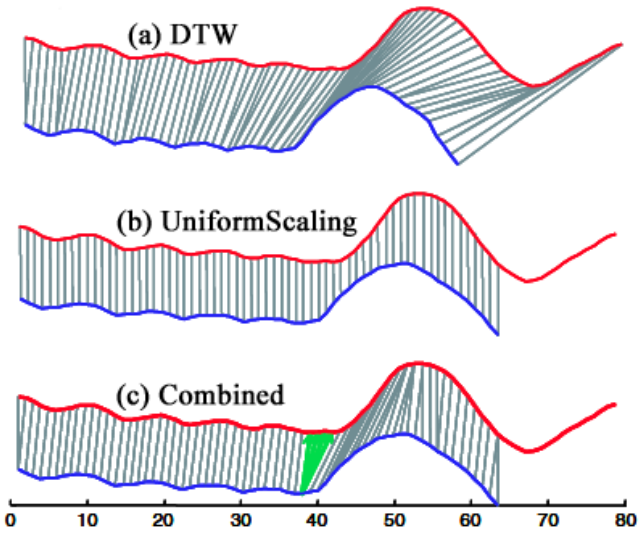
is not clear what should be done with the elements of the longer series that do not have corresponding elements in the shorter series. Second, it is not guaranteed that the first element of a time series marks the start of the gesture. It is possible that one gesture starts after the other with a certain number of "noise" elements at the beginning. To correctly compare the gestures, it is then necessary to shift one of the gestures so that the beginnings are aligned. Third, it is possible that the gestures are performed at different speeds. In many gesture recognition scenarios, we would consider two very similar gestures performed at different speeds to be instances of the same gesture class. It is then necessary to scale one gesture so that it appears to occur at the same speed. Finally, it is possible that the speed of the gestures could vary in different ways at different points. For instance, the first half of one gesture could be relatively fast while the second half of that gesture could be relatively slow. In this scenario, scaling in a uniform way will not produce a successful alignment.

4.2 Dynamic Time Warping

Dynamic Time Warping (DTW) attempts to address the issue of varying speeds in comparing time series by allowing the time series elements of a gesture to be scaled and positioned to produce the best possible match with (lowest possible distance from) another time series. A solution to the problem of aligning time series is discussed in section 4.3. DTW begins by computing a warping matrix consisting of the distances between every item in one series and every item in the other series. Table 1 shows the warping matrix E of two simple time series. Each entry in the warping matrix $E_{i,j}$ is the result of $Dist(A_i, B_j)$. How the distance between entries is calculated is application dependent.

After the warping matrix is calculated, the DTW algorithm selects the element matchings that have the lowest overall distance. These matchings are called the warping path. The warping path describes how the elements of one time series should be scaled and positioned to produce the best possible match with the other series' elements. In Table 1, the first element of series A is matched with the first and second elements of series B. This is analogous to the first step in gesture A being performed twice as quickly as the first step in gesture B. Similarly, the second and third elements of series A are matched with the third element of series B. This is analogous to the second step in gesture A being performed half as quickly as the second step in gesture B. The distance measure returned by DTW given two gestures is the sum of the distances in the warping path. In Table 1, the DTW distance between the two time series is four. [8]

Figure 1: Adding Uniform Scaling to DTW



Part a shows DTW applied to the entirety of both time series. In b and c, the portion of the top line that does not have vertical lines connecting it to the bottom line represents noise data that has been excluded from DTW by Uniform Scaling. Adapted from [2]

4.3 DTW Practical Considerations

For reasonably long time series, computing the entire warping matrix and the warping path can be very computationally costly. Each warping matrix requires $M \times N$ distance calculations where M and N are the lengths of the two time series. In addition, a warping matrix must be calculated between the unclassified gesture and each pre-classified gesture. To reduce the number of calculations required, the approach described in Fu *et al.* [2] attempts to avoid calculating distances that are likely to be very high.

Used by itself, DTW is able to scale and align similar gestures that differ greatly in scale and are significantly misaligned. Unfortunately, this requires calculating a warping matrix that covers every element in both time series. In addition, when one series has significant noise at the beginning or end, DTW will match that noise to points at the beginning or end of the other series (Figure 1.a). To avoid these issues, the approach described in Fu *et al.* [2] first uniformly scales and shifts one gesture to find the best match without DTW and cuts off any time series elements that don't have corresponding elements in the other gesture (Figure 1.b). DTW distance is then calculated between the aligned gestures (Figure 1.c).

After two time series have been aligned by uniform scaling, it is unlikely that any portion of the time series will need to be shifted or scaled by a significant amount. For this reason it is only necessary to calculate elements of the warping matrix $E_{i,j}$ where $|(i - j)| < l$ where l is a predetermined maximum difference between time series positions. The ideal value of l depends on how much local variation in scale is present in the two time series. If l is too large, there will be many unnecessary distance calculations and if l is too small, DTW may miss some warps that would decrease

the final distance measure. For example, if an element in one time series has a very low distance from an element in the other time series 6 positions away, this beneficial warp will be missed if l is 5 or less. By uniform scaling and then selecting an appropriate value of l, the number of required distance calculations between time series elements can be drastically reduced.

5. RECOGNIZING COMPLEX GESTURES WITH HIDDEN MARKOV MODELS

While DTW is able to recognize complex sets of gestures, the gesture class representation used in DTW makes classification computationally costly. Calculating the distances between numerous time series and taking their average is often costly, despite the aforementioned performance optimizations. A Hidden Markov Model (HMM) is a gesture class representation that makes it possible to condense information from multiple pre-classified gesture time series into a single model against which unclassified gestures can be compared. HMMs are commonly employed in gesture recognition [7, 9] as well as in similar fields of study such as speech recognition [5].

5.1 Time Series Quantization

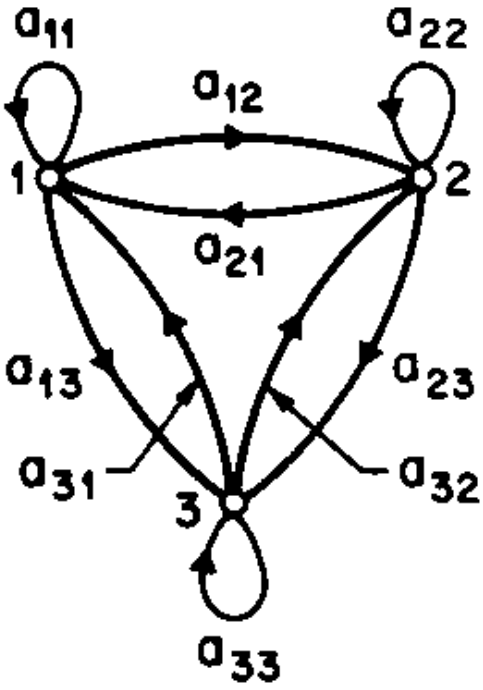
In the gesture recognition scenario we described for DTW, the time series consisted of continuous variables measured at discrete time steps. This means that these time series had a discrete number of elements and that each element contained one or more continuous measurements. As we will soon see, it is difficult to apply HMMs to continuous measurements. For this reason, most HMM applications quantize sensor measurements into a discrete alphabet of events. For example, the three-dimensional continuous output from an accelerometer could be simplified and encoded in a series of three symbols where the symbols correspond to "upward or downward motion" , "rightward or leftward motion", and "forward or backward motion" respectively. This would give us a discrete alphabet of 9 observable events.

5.2 Markov Model Basics

Though Markov Models (MMs) are typically applied to gesture recognition in the context of Hidden Markov Models (HMMs), to simplify our later explanation of HMMs, we will begin by explaining the features of a basic MM and show how it can theoretically be applied to gesture recognition. The first step in describing an MM is defining a list of states S. In the case of Figure 2, $S = \{S_1, S_2, S_3\}$. Each state in the model corresponds to a discrete observable event. There must be a state for all of the events we might observe. Next, we must define a matrix of state transitions A. The dimensions of A are n x n where n is the number of states in the model (the size of S). Each element in A, a_{ij}, corresponds to the probability of transitioning to state j given that we are currently in state i, $P(S_j|S_i)$, in other words, the probability of observing event j given that we just observed event i.[1]

[1]It is worth noting that the probability of transitioning to a particular state only depends on the current state of the model. This is known as the Markov Property. In models that do not exhibit the Markov Property, the probability of transitioning to a particular state can depend not only on the current state of the model, but also on previous states of the model.

Figure 2: A Basic Markov Model



Taken from [5].

Finally, the initial probability of starting in a particular state $P(S_i)$ is given by the vector π, where π_i is the probability of starting in state i.[2]

Given a MM, we would like to calculate the probability of a particular sequence of states (a time series of events) occurring under that model. For example, we would like to calculate $P(S_3, S_3, S_1, S_2, S_3|\pi, A)$. To calculate this probability, we simply multiply together the probability of the initial state followed by the probability of the next state given the previous state and so on until we reach the final state. This is given by $P(S_3) \times P(S_3|S_3) \times P(S_1|S_3) \times P(S_2|S_1) \times P(S_3|S_2)$. The probability of each transition $P(S_j|S_i)$ is given by the corresponding entry in the transition matrix a_{ij}. The probability of this state sequence is then: $\pi_3 \times a_{33} \times a_{31} \times a_{12} \times a_{23}$.

To train an MM to recognize a particular gesture class we need to create a transition matrix such that only series of states corresponding to gestures in that gesture class will receive a relatively high probability. For example, if we wanted to define a gesture class that only contains gestures with the state sequence $\{S_1, S_2, S_3\}$, we would use this transition matrix A and initial probability vector π

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\pi = \{1, 0, 0\}$$

Given this model, the state sequence $\{S_1, S_2, S_3\}$ will receive a probability of 1 while all other state sequences will receive a

[2]The rows of A must sum to 1 and π must sum to 1.

probability of 0. This is a very strict MM. In most cases, the MM will be set up to allow some variation in the unclassified state sequences. For example if we change the transition matrix to

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0.2 & 0.8 \\ 0 & 0 & 0 \end{bmatrix}$$

the model will allow for state sequences that remain in S_2 for more than one time step, but the calculated probability of that state sequence will decrease the more instances of S_2 are present in the time series. For example

$$\begin{aligned} P(S_1, S_2, S_2, S_3) &= P(S_1) \times P(S_2|S_1) \times P(S_2|S_2) \times P(S_3|S_2) \\ &= \pi_1 \times a_{12} \times a_{22} \times a_{23} \\ &= 1 \times 1 \times 0.2 \times 0.8 \\ &= 0.16 \end{aligned}$$

and $P(S_1, S_2, S_2, S_2, S_3) = 1 \times 1 \times 0.2 \times 0.2 \times 0.8 = 0.032$. The probability returned reflects how closely the gesture represented by the state sequence matches the gesture class represented by the model. It is worth noting that this probability does not indicate the probability that the state sequence corresponds to the gesture. Probabilities returned by all but the simplest MMs are very small and are only useful for comparison with probabilities returned by other MMs applied to the same state sequence. For example, given a single time series, a returned probability of 0.0001 could indicate a close match if all other MMs returned significantly smaller probabilities.

5.3 Hidden Markov Models

Using Markov Models, we are severely limited in our ability to model gestures. Because the probability of observing an event can only be dependent on the previous event, many complex event sequences can not be trained into MMs. Hidden Markov Models are an extension of MMs with expanded recognition capability. In a HMM, there is an underlying Markov Model, but this MM is defined not in terms of events coming from our sensing technology, but in terms of the idealized steps of the gesture. This model is considered hidden because we can not directly observe the steps of the gesture from the sense data. To connect the states of the underlying MM and the events we observe, we create a set of observable value probability distributions B. B can be created in a number of ways. If the raw sense data has been quantized into events, B_i is a vector where B_{ij} represents the probability of observing event j in state i $\mathrm{P}(j|S_i)$. If the raw sense data has not been quantized and is instead a continuous measurement, B_i is a function that returns the probability of observing a particular continuous measurement in state i.

To produce a HMM to recognize a gesture class, we not only have to specify S, A, and π, we also must specify a set of observable value probability distributions B, one for each state in S. We would like to find weightings of A, B, and π that maximize the probabilities returned when that HMM is applied to members of a particular gesture class. This can be accomplished using the Baum-Welch method. The method takes in a set of pre-classified gestures and produces a set of weightings that maximize the probability returned for those gestures. If the pre-classified gestures are sufficiently representative of the gesture class, the resulting HMM will also return relatively high probabilities for unclassified gestures that are members of that gesture class. A more detailed explanation of the Baum-Welch method can be found in Rabiner [5]. Given an HMM trained to recognize a particular gesture class, calculating the probability of an unclassified gesture is much less straightforward than doing so with a MM. This calculation involves not only the probability of a particular state sequence, but also the probability of observing each event in it's corresponding state. In addition, we can not simply follow the event sequence given to us; instead, we must produce our own state sequence that maximizes the returned probability. For example, given an event sequence, we could remain in S_1 for each event. It is more likely, however that we can get a greater probability by transitioning to different states to consume different events. To illustrate how HMM probabilities are calculated and how we might find the optimal state sequence for a particular event sequence, we will imagine a scenario where only 2 events can be observed: $\mathcal{U}$ corresponds to an upward motion and $\mathcal{D}$ corresponds to a downward motion. Given a two-state MM:

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\pi = \{1, 0\}$$

we will now describe an observable value probability distribution for each state.

	$\mathcal{U}$	$\mathcal{D}$
B_1	0	1
B_2	1	0

For this gesture class, only a downward motion is allowed in S_1 and only an upward motion is allowed in S_2. We will now try to find the optimal state sequence given the event sequence: $\{\mathcal{D},\mathcal{U}\}$. We could try remaining in S_1 for both events (S_1, S_1). This probability is

$$\begin{aligned} P(\mathcal{D},\mathcal{U}) &= P(S_1) \times P(\mathcal{D}|S_1) \times P(S_1|S_1) \times P(\mathcal{U}|S_1) \\ &= \pi_1 \times B_{1\mathcal{D}} \times a_{11} \times B_{1\mathcal{U}} \\ &= 1 \times 1 \times 0 \times 0 \\ &= 0 \end{aligned}$$

This is not likely the correct state sequence. Instead we could try starting in S_1 for the first event, then transitioning to S_2 for the second event (S_1, S_2). This probability is

$$\begin{aligned} P(\mathcal{D},\mathcal{U}) &= P(S_1) \times P(\mathcal{D}|S_1) \times P(S_2|S_1) \times P(\mathcal{U}|S_2) \\ &= \pi_1 \times B_{1\mathcal{D}} \times a_{12} \times B_{2\mathcal{U}} \\ &= 1 \times 1 \times 1 \times 1 \\ &= 1 \end{aligned}$$

We have likely found the optimal state sequence for the observation sequence. Though this is obviously a rather contrived example, it illustrates how HMM probabilities are calculated. At each time step, we take into account the probability of arriving at the current state given the previous state as well as the probability of observing the current event in the current state.

For longer time series with more observable events, the task of finding an optimal state sequence becomes extremely complicated. To find the optimal state sequence we could try all possible state sequences and choose the one that produces the highest probability given the observation sequence. Unfortunately, this approach has a time complexity of $O(s^n)$

where s is the number of states in the model and n is the length of the observation sequence. The Viterbi algorithm as described in Rabiner [5] is able to find an optimal state sequence with a time complexity of $O(n \times s^2)$.

5.4 HMMs in Practice

When applying HMMs to real-world scenarios, their design typically differs from the theoretical description. While in theory, the states of the underlying MM should correspond to the steps of the gesture, in practice, defining those steps is not straightforward. In addition, HMMs with more states tend to return lower probability values, so to compare the output of a set of HMMs with different numbers of states, a method for normalizing the returned probabilities is required. For these reasons, most HMM gesture recognition implementations use the same number of states for each HMM. While one would think that having a set number of states would be problematic as different gestures have different numbers of steps, in practice the Baum-Welch method can usually produce effective recognizers as long as the number of states is within a reasonable range, typically between 2 and 8. Because of this disconnect between the theoretical and practical function of states, mapping the weightings of A and B produced by the Baum-Welch method to how a human would describe a gesture class is rarely straightforward. Most often, researchers do not attempt to analyze exactly how a particular HMM recognizes a gesture class. Instead, they focus on selecting the optimal number of states for all the HMMs and producing useful pre-classified gestures for training.

5.5 Comparison with DTW

The recognition accuracy of DTW and HMMs is heavily dependent on a number of factors including the quality of the training examples and the distinctness of the the gesture classes. Despite this, some general trends have been observed. Kogan *et al.* [4] note that while the accuracy of the algorithms tends to improve with more pre-classified gestures, DTW can significantly outperform HMMs when the number of pre-classified gestures is very low (5 or less). With a sufficiently large number of noise-free pre-classified gestures, the accuracy of the two methods is approximately equal. If there is considerable noise in the pre-classified gestures, HMMs outperform DTW. For an in-depth analysis of these algorithm's performance in varying recognition scenarios, see Kogan *et al.* [4].

6. CONCLUSION

In this paper we have described three common approaches to gesture recognition: namely, Linear Evaluation Functions, Dynamic Time Warping, and Hidden Markov Models. We have described what is common between the approaches and how they differ. We have also briefly highlighted the advantages and disadvantages of each approach. With this basic introduction to the field, one can see how the seemingly intelligent feat of gesture recognition can be accomplished fairly successfully with statistical and mathematical models. The three approaches described in this paper are among the most commonly employed, but they are by no means the only approaches being researched today. The basic gesture recognition structure we described in section 2 provides a basic framework for thinking about gesture recognition to which a broad range of algorithms can be applied.

7. REFERENCES

[1] D. Ashbrook and T. Starner. Magic: A motion gesture design tool. In *Proceedings of the 28th international conference on Human factors in computing systems*, CHI '10, pages 2159–2168, New York, NY, USA, 2010. ACM.

[2] A. W.-C. Fu, E. Keogh, L. Y. Lau, C. A. Ratanamahatana, and R. C.-W. Wong. Scaling and time warping in time series querying. *The VLDB Journal*, 17(4):899–921, July 2008.

[3] L. Geurts, A. Van Woensel, and V. V. Abeele. No sweat, no fun: Large-gesture recognition for computer games. In *Proceedings of the 4th International Conference on Fun and Games*, FnG '12, pages 109–112, New York, NY, USA, 2012. ACM.

[4] J. Kogan and D. Margoliash. Automated recognition of bird song elements from continuous recordings using dynamic time warping and hidden markov models: A comparative study. *Journal of the Acoustical Society of America.*, 103(4), Apr. 1998.

[5] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In A. Waibel and K.-F. Lee, editors, *Readings in speech recognition*, pages 267–296. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[6] D. Rubine. Specifying gestures by example. *SIGGRAPH Comput. Graph.*, 25(4):329–337, July 1991.

[7] T. Schlömer, B. Poppinga, N. Henze, and S. Boll. Gesture recognition with a Wii controller. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, TEI '08, pages 11–14, New York, NY, USA, 2008. ACM.

[8] P. Senin. Dynamic time warping algorithm review. Technical Report CSDL-08-04, Department of Information and Computer Sciences, University of Hawaii, Honolulu, Hawaii 96822, Dec. 2008.

[9] T. Westeyn, H. Brashear, A. Atrash, and T. Starner. Georgia Tech gesture toolkit: Supporting experiments in gesture recognition. In *Proceedings of the 5th international conference on Multimodal interfaces*, ICMI '03, pages 85–92, New York, NY, USA, 2003. ACM.