Nearest Neighbor Search Methods for Handshape Recognition

Michalis Potamias[1] and Vassilis Athitsos[2]
[1] Computer Science Department, Boston University
[2] Computer Science and Engineering Department, University of Texas at Arlington

ABSTRACT

Gestures are an important modality for human-machine communication, and robust gesture recognition can be an important component of intelligent homes and assistive environments in general. An important aspect of gestures is handshape. Handshapes can hold important information about the meaning of a gesture, for example in sign languages, or about the intent of an action, for example in manipulative gestures or in virtual reality interfaces. At the same time, recognizing handshape can be a very challenging task, because the same handshape can look very different in different images, depending on the 3D orientation of the hand and the viewpoint of the camera. In this paper we examine a database approach for handshape classification, whereby a large database of tens of thousands of images is used to represent the wide variability of handshape appearance. Efficient and accurate indexing methods are important in such a database approach, to ensure that the system can match every incoming image to the large number of database images at interactive times. In this paper we examine the use of embedding-based and hash table-based indexing methods for handshape recognition, and we experimentally compare these two approaches on the task of recognizing 20 handshapes commonly used in American Sign Language (ASL).

Categories and Subject Descriptors

I.4.8 [**Scene Analysis**]: Object Recognition; H.3.1 [**Content Analysis and Indexing**]: Indexing methods; H.2.8 [**Database Applications**]: Data Mining; H.2.4 [**Systems**]: Multimedia Databases

Keywords

gesture recognition, hand pose estimation, embeddings, hash-based indexing, BoostMap, distance-based hashing, DBH

1. INTRODUCTION

Gestures are an important modality for human-machine communication, and robust gesture recognition can be an important component of intelligent homes and assistive environments in general. An important aspect of gestures is handshape. Handshapes can hold important information about the meaning of a gesture, for example in sign languages, or about the intent of an action, for example in manipulative gestures or in virtual reality interfaces.

Recognizing handshape can be a very challenging task, because the same handshape can look very different in different images, depending on the 3D orientation of the hand and the viewpoint of the camera. In this paper we examine a database approach for handshape classification, whereby a large database of tens of thousands of images is used to represent the wide variability of handshape appearance. A key advantage of the database approach is that it provides a very natural way to capture the nonparametric distribution that characterizes the appearance of each handshape class. Furthermore, databases containing tens or hundreds of thousands of images can be easily generated overnight using off-the-shelf computer graphics software.

Efficient and accurate indexing methods are important in such a database approach. Given an input image, the system identifies the most similar image in the database, so that the handshape in the input image gets classified according to the handshape label of the best matching database image. This best matching database image needs to be identified fast enough to allow the system to be used in an interactive environment. At the same time, this database retrieval task can be very challenging, for the following reasons:

- The similarity measures that are most meaningful in comparing hand images are typically non-Euclidean, nonmetric, and computationally expensive. Examples of such nonmetric distance measures are the chamfer distance [6], shape context matching [7, 31], and distance measures based on the Viterbi algorithm [31].

- The majority of database indexing methods are designed for Euclidean distance measures or metric distance measures (i.e., distance measures that obey the reflexivity, symmetry, and triangle inequality properties). Thus a relatively small number of indexing methods are available for the nonmetric distance measures typically used for comparing hand images.

In this paper we examine the use of recently proposed embedding-based and hash table-based indexing methods for handshape recognition. In particular, we consider the BoostMap embedding method [3] and Distance-Based Hashing (DBH) [5]. We discuss how to apply those methods for efficient retrieval of hand images, and we compare the performance of both methods on the task recognizing 20 handshapes commonly used in American Sign Language. The main conclusion is that both methods offer orders-of-magnitude speedups compared to the naive brute-force method of comparing the input image to every single database image. Overall, the experiments demonstrate that the database approach we describe is a scalable and feasible approach for handshape recognition from arbitrary camera viewpoints.

2. RELATED WORK

Computer vision systems that estimate handshape under arbitrary 3D orientations typically do it in the context of tracking [18,

PETRA 2008 Athens, Greece
Copyright 2008.

24, 25, 30, 38]. In that context, the pose can be estimated at the current frame as long as the system knows the pose at the previous frame. Since such trackers rely on knowledge about the previous frame, they need to be manually initialized, and cannot recover when they lose the track. The method described in this paper can be used (among other things) to automate the initialization and error recovery of a hand tracker.

A regression system that estimates hand pose from a single image is described in [26]. However, that method assumes that the hand silhouette is correctly identified in the input image, whereas such precise hand detection is often unrealistic to assume in a real-world application. Another regression method is presented at [13], but that method requires that the hand be simultaneously visible from multiple cameras. The database approach described here has the advantage that it only requires a single camera, and it can tolerate a certain amount of imprecision in hand detection; we still require the location of the hand to be given as an input to our system, but we do not require precise separation of the hand silhouette from the background.

Another family of methods for hand shape classification are appearance-based methods, like [15, 37]. Such methods are typically limited to estimating 2D hand pose from a limited number of viewpoints. In contrast, the method described in this paper can handle arbitrary viewpoints.

Our system uses the chamfer distance [6] to compute the similarity between the input hand image and database images. The main focus of this paper is on identifying efficient indexing methods for speeding up the task of finding, given the input image, the best matching database images. Various methods have been employed for speeding up nearest neighbor retrieval. Comprehensive reviews on the subject include [8, 19, 20]. A large amount of work focuses on efficient nearest neighbor retrieval in multidimensional vector spaces using an L_p metric, e.g., [22, 32, 36]. However, that family of approaches is not applicable in our setting, since the chamfer distance (i.e., the distance measure that we use for comparing hand images) is not an L_p measure.

A number of nearest neighbor methods can be applied for indexing arbitrary metric spaces; the reader is referred to [20] for surveys of such methods. As an example, VP-trees [39] and metric trees [33] hierarchically partition the database into a tree structure by splitting, at each node, the set of objects based on their distances to pivot objects. However, while such methods can offer theoretical guarantees of performance in metric spaces, the chamfer distance used in our system is nonmetric, and so are other measures typically used for comparing hand images to each other, such as shape context matching [7, 31], and distance measures based on the Viterbi algorithm [31].

In domains with a computationally expensive distance measure, significant speed-ups can be obtained by embedding objects into another space with a more efficient distance measure. Several methods have been proposed for embedding arbitrary spaces into a Euclidean or pseudo-Euclidean space [3, 4, 9, 14, 21, 35]. These methods are indeed applicable to our setting. In this paper we focus on the BoostMap embedding method [3] and we show that this method can be successfully employed for efficient matching of hand images.

Locality Sensitive Hashing (LSH) is an approximate nearest neighbor method that is based on hash tables. LSH has been shown theoretically to scale well with the number of dimensions and has produced good results in practice [17, 28]. However, LSH cannot be applied to arbitrary distance measures, and there is no existing method that allows applying LSH to the chamfer distance. An alternative hash-based method that can be applied to arbitrary distance measures, such as the chamfer distance, is Distance-Based Hashing (DBH) [5]. In this paper we explore the usage of DBH for efficiently identifying the best matching database image for the input hand image, and we compare the performance of DBH to that of the BoostMap embedding method.

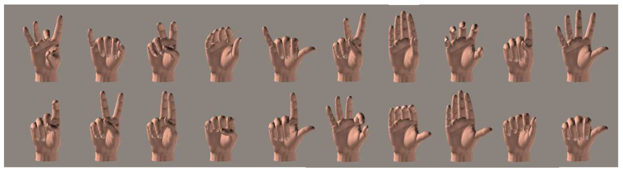
Figure 1: The 20 handshapes used in the ASL handshape dataset.

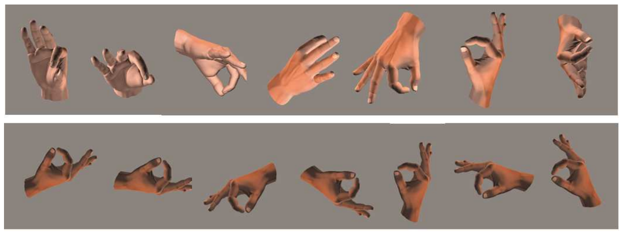
Figure 2: Examples of different appearance of a fixed 3D hand shape, obtaining by altering camera viewpoint and image plane rotation. Top: the ASL "F" handshape rendered from seven different camera viewpoints. Bottom: the ASL "F" handshape rendered from a specific camera viewpoint, using seven different image plane rotations.

3. HANDSHAPE RECOGNITION USING A DATABASE

Our goal is to have a system that can recognize a set of different handshapes, such as the 20 handshapes shown on Fig. 1. We want this system to operate on single images, as opposed to entire video sequences, or images obtained simultaneously for multiple cameras. We need to specify up front that, in a real-world system, reliable recognition of handshapes of arbitrary 3D orientation from a single image is beyond the current state of the art. At the same time, a system that operates on a single image, even if it has a relatively low classification accuracy, can be immensely useful in identifying a relatively small set of likely hypotheses. Such a set of hypotheses can subsequently be refined:

- using a hand tracker [25, 18, 24, 29, 30, 38],
- using domain-specific knowledge, such as ASL linguistic constraints, or
- using knowledge of a specific protocol for human-computer communication, that can place constraints on the current handshape based on the current communication context.

A key challenge in reliable handshape recognition in an intelligent home setting, or an assistive environment setting, is that the same handshape can look very different in different images, depending on the 3D orientation of the hand with respect to the camera (Fig. 2). Using a large database of hand images is a natural way to address this wide variability of the appearance of a single handshape. Since handshape appearance depends on 3D orientation, we can densely sample the space of all possible 3D orientations, and include a database image for every handshape in every one of the sampled 3D orientations.

In our system, we include 20 different handshapes (Fig. 1). Those 20 handshapes are all commonly used in American Sign Language (ASL). For each handshape, we synthetically generate a total of 4,032 database images that correspond to different 3D orientations of the hand. In particular, the 3D orientation depends on the viewpoint, i.e., the camera position on the surface of a viewing sphere centered on the hand, and on the image plane rotation. We sample 84 different viewpoints from the viewing sphere, so that viewpoints are approximately spaced 22.5 degrees apart. We also sample 48 image plane rotations, so that rotations are spaced 7.5 degrees apart. Therefore, the total number of images is 80,640 images, i.e., 20 handshapes $\times$ 84 viewpoints $\times$ 48 image plane rotations. Figure 2 displays example images of a handshape in different viewpoints and different image plane rotations. Each image is normalized to be of size 256×256 pixels, and the hand region in the image is normalized so that the minimum enclosing circle of the hand region is centered at pixel $(128, 128)$, and has radius 120. All database images are generated using computer graphics, and in particular using the Poser 5 software [12]. It takes less than 24 hours to generate these thousands of images. Image generation is a script-based automated process.

4. THE CHAMFER DISTANCE

Given an input image, the system has to identify the database images that are the closest to the input. In our system we measure distance between edge images, because edge images tend to be more stable than intensity images with respect to different lighting conditions. Examples of hand images and corresponding edge images are shown on Fig. 3.

The chamfer distance [6] is a well-known method to measure the distance between two edge images. Edge images are represented as sets of points, corresponding to edge pixel locations. Given two edge images, X and Y, the chamfer distance $D(X, Y)$ is:

$$D(X,Y) = \frac{1}{|X|}\sum_{x\in X}\min_{y\in Y}\|x-y\| + \frac{1}{|Y|}\sum_{y\in Y}\min_{x\in X}\|y-x\| \ , \quad (1)$$

where $\|a - b\|$ denotes the Euclidean distance between two pixel locations a and b. $D(X, Y)$ penalizes for points in either edge image that are far from any point in the other edge image. Fig. 4 shows an illustration of the chamfer distance.

The chamfer distance operates on edge images. The synthetic images generated by Poser can be rendered directly as edge images by the software. For the test images we simply apply the Canny edge detector [11].

On an AMD Athlon processor running at 2.0GHz, we can compute on average 715 chamfer distances per second. Consequently, finding the nearest neighbors of each test image using brute force search, which requires computing the chamfer distances between the test image and each database image, takes about 112 seconds. Taking 112 seconds to match the input image with the database is clearly too long for an interactive application. The need for efficiency motivates our exploration of database indexing methods, such as embeddings and hash tables. We now proceed to describe how such methods can be applied.

5. EMBEDDING-BASED RETRIEVAL

In our application, calculating the chamfer distance between the input image and all database images takes too long (almost two minutes) to be used in interactive applications. However, we can

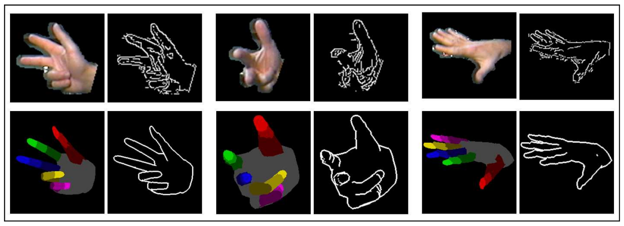

Figure 3: Examples of real and synthetic hand images and their corresponding edge images.

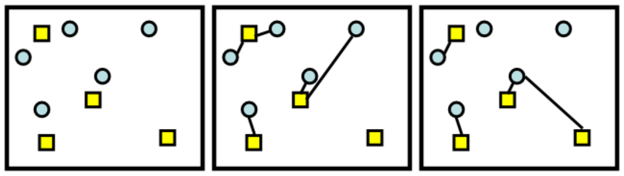

Figure 4: An example of the chamfer distance. The left image shows two sets of points: points in the first set are shown as circles, and points in the second set are shown a squares. The middle image shows a link between each circle and its closest square. The circle-to-square directed chamfer distance is the average length of those links. The right image shows a link between each square and its closest circle. The square-to-circle chamfer distance is the average length of those links. The chamfer distance (also known as *undirected chamfer distance*) between squares and circles is the sum of the two directed distances.

obtain an efficient approximation of the chamfer distance by embedding edge images into a vector space. Using such an embedding we can drastically speed up classification time, without any noticeable decrease in classification accuracy. In this section we discuss how such embeddings can be constructed.

5.1 Background: Lipschitz Embeddings

Embeddings of arbitrary spaces into a vector space are a general approach for speeding up nearest neighbor retrieval. Let $\mathbb{X}$ be a set of objects (the set of the edge images of hands in our case, see Fig. 3 for examples), and $D(X_1, X_2)$ be a distance measure between objects $X_1, X_2 \in \mathbb{X}$. In our case, D is the chamfer distance (Eq. 1). An embedding $F : \mathbb{X} \rightarrow \mathbb{R}^d$ is a function that maps objects from X into the d-dimensional real vector space $\mathbb{R}^d$, where distances are typically measured using an L_p or weighted L_p measure, denoted as D'. Such embeddings are useful when it is computationally expensive to evaluate distances in $\mathbb{X}$, and it is more efficient to map points of $\mathbb{X}$ to vectors and compute some L_p distance between those vectors.

Given an object $X \in \mathbb{X}$, a simple 1D embedding $F^R : \mathbb{X} \rightarrow \mathbb{R}$ can be defined as follows:

$$F^R(X) = D(X, R) \ . \quad (2)$$

The object R that is used to define F^R is typically called a *reference object* or a *vantage object* [19]. A multidimensional embedding $F : \mathbb{X} \rightarrow \mathbb{R}^d$ can be constructed by concatenating such 1D embeddings: if $F_1, \ldots, F_d$ are 1D embeddings, we can define a d-dimensional embedding F as $F(X) = (F_1(X), \ldots, F_d(X))$.

The basic intuition behind such embeddings is that two objects that are close to each other typically have similar distances to all other objects. An everyday example that illustrates this property is looking at distances between cities. The distance from New York to Boston is about 240 miles, and the distance from New York to Los Angeles is about 2800 miles. Suppose that we did not know these two distances. Furthermore, suppose that someone gave us, for 100 towns spread across the United States, their distances to New York, Boston and Los Angeles. What would that information tell us about the distances from New York to Boston and from New York to Los Angeles?

First we would notice that the distance from each town to New York is always within 240 miles or less of the distance between that town and Boston. On the other hand, there are some towns, like Lincoln, Nebraska, whose distances from Los Angeles and New York are very similar, and some towns, like Sacramento, whose distances to Los Angeles and New York are very different (Sacramento-Los Angeles is 400 miles, Sacramento-New York is 2800 miles). Given these distances, we could deduce that, most likely, New York is a lot closer to Boston than it is to Los Angeles.

In our handshape recognition application, suppose that we have chosen a set of d database edge images $R_1, R_2, ..., R_k$ as reference objects. Then, we can define a function F, mapping the space of edge images to $\mathbb{R}^d$ as follows:

$$F(X) = (D(X, R_1), D(X, R_2), ..., D(X, R_d)) . \qquad (3)$$

where D is the chamfer distance, defined in Equation 1, and X is an edge image. The function F turns out to be a special case of Lipschitz embeddings [9, 23].

We define the *approximate chamfer distance* D' between two edge images X_1 and X_2 to be the L_1 distance between $F(X_1)$ and $F(X_2)$:

$$D'(A, B) = \sum_{i=1}^{d} |D(X_1, R_i) - D(X_2, R_i)| . \qquad (4)$$

The actual value of $D'(A, B)$ is not necessarily similar in scale to the value $D(A, B)$. However, $D'(A, B)$ is an approximation of $D(A, B)$ in the sense that, when $D(A, B)$ is much smaller than $D(A, G)$, then we also expect $D'(A, B)$ to be smaller than $D'(A, G)$. The intuition is, again, that if A and B are close to each other, then they will also have relatively similar distances to each of the R_i's.

The time complexity of computing the approximate distance D' between an edge image X and U database edge images is $O(dn \log n + Ud)$, where n is the max number of edge pixels in any edge image and d is the dimensionality of the embedding. In particular, it takes $O(dn \log n)$ time to compute $F(X)$, i.e., to compute the d chamfer distances between the edge image and each of the d reference objects, and it takes $O(Ud)$ time to compute the L_1 distance between $F(X)$ and the embeddings of all database images (which just need to be precomputed once, off-line, and stored in memory). On the other hand, computing the chamfer distance C between X and all database images takes $O(Un \log n)$ time. The complexity savings are substantial when d is much smaller than U. In our system it takes on average 112 seconds to compute the chamfer distances between the input image and all database images (for test and database images of size 256x256). In contrast, for $d = 100$, it takes 0.14 seconds to compute the corresponding approximate distances D'.

5.2 BoostMap Embeddings

A simple way to define embeddings for our purposes, i.e., for efficient matching of hand images, is to apply Eq. 3 for some reasonable embedding dimensionality d (values between 20 and 100 typically work well in practice), and using d reference objects R_i chosen randomly from the database. However, we can significantly optimize embedding quality using tools available from the machine learning community. In particular, embedding optimization can be casted as the machine learning problem of optimizing a binary classifier, and boosting methods such as AdaBoost [27] can be employed for embedding optimization [3], as described in the next paragraphs.

Suppose we have an embedding F with the following property: for any $Q, A, B \in \mathbb{X}$ (where $\mathbb{X}$ is our space of edge images of hands), if Q is closer (according to the chamfer distance) to A than to B, then $F(Q)$ is closer to $F(A)$ than to $F(B)$. We can easily derive that F would also have the following property: for every input image Q, if A is the nearest neighbor of Q in the database, then $F(A)$ is the nearest neighbor of $F(Q)$ among the embeddings of all database objects. Such an embedding would lead to perfectly accurate nearest neighbor retrieval.

Finding such a perfect embedding is usually impossible. However, we can try to construct an embedding that, as much as possible, tries to behave like a perfect embedding. In other words, we want to construct an embedding in a way that maximizes the fraction of triples (Q, A, B) such that, if Q is closer to A than to B, then $F(Q)$ is closer to $F(A)$ than to $F(B)$.

More formally, using an embedding F we can define a classifier $\tilde{F}$, that estimates (sometimes wrongly) for any three objects Q, A, B if Q is closer to A or to B. $\tilde{F}$ is defined as follows:

$$\tilde{F}(Q, A, B) = \|F(Q) - F(B)\|_1 - \|(F(Q) - F(A)\|_1 , \qquad (5)$$

where $\|X, Y\|_1$ is the L_1 distance between X and Y. A positive value of $\tilde{F}(Q, A, B)$ means that F maps Q closer to A than to B, and can be interpreted as a "prediction" that Q is closer to A than to B in the original space X. If this prediction is always correct, then F perfectly preserves the similarity structure of X.

Simple 1D embeddings, like the one defined in Eq. 2, are expected to behave as *weak classifiers*, i.e. classifiers that may have a high error rate, but at least give answers that are not as bad as random guesses (random guesses are wrong 50% of the time). Given many weak classifiers, a well-studied problem in machine learning is how to combine such classifiers into a single, strong classifier, i.e., a classifier with a low error rate. A popular choice is AdaBoost [27], which has been successfully applied to several domains in recent years.

The BoostMap algorithm [3] uses AdaBoost to construct an embedding. The input to AdaBoost is a large set of randomly picked 1D embeddings (i.e., embeddings defined by applying Eq. 2 using reference objects R picked randomly from our database), and a large set of training triples (Q, A, B) of objects, for which we know if Q is closer to A or to B (closer according to the chamfer distance, in our case). The output of AdaBoost is a classifier $H = \sum_{j=1}^{d} \alpha_j \tilde{F}_j$, where each $\tilde{F}_j$ is the weak classifier associated with a 1D embedding F_j. If AdaBoost has been successful, then H has a low error rate.

Using H, we can easily define a high-dimensional embedding F_{out} and a distance measure D' with the following property: for any triple (Q, A, B), if Q is closer to A than to B, H misclassifies that triple if and only if, according to distance measure D' (i.e., the L_1 distance measure in the embedding space) $F_{\text{out}}(Q)$ is closer to $F_{\text{out}}(B)$ than to $F_{\text{out}}(A)$. We define F_{out} and D' as follows:

$$F_{\text{out}}(x) = (F_1(x), ..., F_d(x)) . \qquad (6)$$

$$D'(F_{\text{out}}(x), F_{\text{out}}(y)) = \sum_{j=1}^{d} (\alpha_j |F_j(x) - F_j(y)|) . \qquad (7)$$

It is easy to prove that H and $F_{\rm out}$ fail on the same triples [3]. Therefore, if AdaBoost has successfully produced a classifier H with low error rate, then $F_{\rm out}$ inherits the low error rate of H.

6. DISTANCE-BASED HASHING

In this section we describe Distance-Based Hashing (DBH), a method for applying hash-based indexing in arbitrary spaces and distance measures, introduced in [5]. In order to make our method applicable to arbitrary spaces (such as the space of hand images under the chamfer distance measure), a key requirement is to use the distance measure as a black box. Therefore, the definition of the hash functions should only depend on distances between objects. To keep the method general, no additional assumptions are made about the distance measure. In particular, the distance measure is *not* assumed to have Euclidean or metric properties.

In existing literature, several methods have been proposed for defining functions that map an arbitrary space $(\mathbb{X}, D)$ into the real line $\mathbb{R}$. An example is the pseudo line projections proposed in [14]: given two arbitrary objects $X_1, X_2 \in \mathbb{X}$, we define a "line projection" function $F^{X_1,X_2} : \mathbb{X} \rightarrow \mathbb{R}$ as follows:

$$F^{X_1,X_2}(X) = \frac{D(X,X_1)^2 + D(X_1,X_2)^2 - D(X,X_2)^2}{2D(X_1,X_2)} \ . \quad (8)$$

If $(\mathbb{X}, D)$ is a Euclidean space, then $F^{X_1,X_2}(X)$ computes the projection of point X on the unique line defined by points X_1 and X_2. If $\mathbb{X}$ is a general non-Euclidean space, then $F^{X_1,X_2}(X)$ does not have a geometric interpretation. However, as long as a distance measure D is available, F^{X_1,X_2} can still be defined and provides a simple way to project $\mathbb{X}$ into $\mathbb{R}$.

We should note that the family of functions defined using Equation 8 is a very rich family. Any pair of objects defines a different function. Given a database $\mathbb{U}$ of n objects, we can define about $n^2/2$ unique functions by applying Equation 8 to pairs of objects from $\mathbb{U}$.

Functions defined using Equation 8 are real-valued, whereas hash functions need to be discrete-valued. We can easily obtain discrete-valued hash functions from F^{X_1,X_2} using thresholds $t_1, t_2 \in \mathbb{R}$:

$$F^{X_1,X_2}_{t_1,t_2}(X) = \begin{cases} 0 & \text{if } F^{X_1,X_2}(X) \in [t_1, t_2] \ . \\ 1 & \text{otherwise} \ . \end{cases} \quad (9)$$

In practice, t_1 and t_2 should be chosen so that $F^{X_1,X_2}_{t_1,t_2}(X)$ maps approximately half the objects in $\mathbb{X}$ to 0 and half to 1, so that we can build balanced hash tables. We can formalize this notion by defining, for each pair $X_1, X_2 \in \mathbb{X}$, the set $\mathbb{V}(X_1, X_2)$ of intervals $[t_1, t_2]$ such that $F^{X_1,X_2}_{t_1,t_2}(X)$ splits the space in half:

$$\mathbb{V}(X_1,X_2) = \{[t_1,t_2] | \Pr_{X\in\mathbb{X}}(F^{X_1,X_2}_{t_1,t_2}(X) = 0) = 0.5\} \ . \quad (10)$$

Note that, for a set of n objects, there are n ways to split those objects into two equal-sized subsets (if n is even) based on the choice of $[t_1, t_2] \in \mathbb{V}(X_1, X_2)$. One of several alternatives is to choose an interval $[t_1, \infty]$ such that $F^{X_1,X_2}(X)$ is less than t_1 for half the objects $X \in \mathbb{X}$. Another alternative is, for example, to choose an interval $[t_1, t_2]$ such that, using F^{X_1,X_2}, one sixth of the objects in X are mapped to a value less than t_1 and two sixths of the objects are mapped to a value greater than t_2. The set $\mathbb{V}(X_1, X_2)$ includes intervals for all these possible ways to split X into two equal subsets.

Using the above definitions, we are now ready to define a family $\mathcal{H}_{\rm DBH}$ of hash functions for an arbitrary space $(\mathbb{X}, D)$:

$$\mathcal{H}_{\rm DBH} = \{F^{X_1,X_2}_{t_1,t_2} | X_1, X_2 \in \mathbb{X}, [t_1,t_2] \in \mathbb{V}(X_1,X_2)\} \ . \quad (11)$$

Using random binary hash functions h_{ij} sampled (with replacement) from $\mathcal{H}_{\rm DBH}$ we can define k-bit hash functions g_i as follows:

$$g_i(X) = (h_{i1}(X), h_{i2}(X), \ldots, h_{ik}(X)) \ . \quad (12)$$

This way, indexing and retrieval can be performed as in Locality Sensitive Hashing (LSH)[16], by:

- Choosing parameters k and l.
- Constructing l k-bit hash tables, and storing each database object to the appropriate l buckets.
- Comparing the input image with the database images found in the l hash table buckets that the input image is mapped to.

Appropriate values for k and l can be computed using a validation set of test objects (in our case, a validation set of hand images) [5].

While there are important similarities between DBH and LSH [16], there are also important differences. Applying the LSH framework to a specific space and distance measure requires identifying a locality sensitive family [16]. Such families have been identified for certain spaces, such as vector spaces with L_p metrics [2, 16], or strings with a substitution-based distance measure [1, 10]. In Euclidean space $\mathbb{R}^d$, the time complexity of retrieval using LSH is linear in the dimensionality d and sublinear in the number n of database objects [2].

At the same time, LSH cannot be applied to an arbitrary non-Euclidean distance measure (such as the chamfer distance), unless a locality sensitive family of hash functions is identified. There exists no general way of constructing/identifying locality sensitive families for arbitrary distance measures. In DBH, a rich family of hashing functions is constructed using Eq. 11. That family of functions is not locality sensitive, and therefore the LSH theoretical analysis is not applicable for DBH. However, the key advantage of DBH is that the family of hashing functions is defined in a domain-independent way, and can be constructed for any space and any distance measure. Thus, DBH can be easily applied to index the chamfer distance.

7. FILTER-AND-REFINE RETRIEVAL

Sections 5.2 and 6 have described two different approaches, BoostMap and DBH, for efficient nearest neighbor search. However, what these methods have in common is that each of them provides an efficient way to identify a relatively small set of candidate nearest neighbors out of the entire database of hand images. In this section we discuss how to implement an end-to-end retrieval system using each of these methods. Essentially, both methods are natural fits for the filter step of the well-known filter-of-refine retrieval framework [19], which works as follows:

- **Offline preprocessing step:** compute and store information about database objects that is useful for indexing. For BoostMap, this step involves computing the embeddings of all database objects. For DBH, preprocessing involves constructing the l hash tables and storing, for each database object, a pointer to that object in the appropriate bin for each of the l hash tables.
- **Mapping step:** given an input image Q, compute the embedding of Q (for the BoostMap method), or compute the l hash keys that Q corresponds to (for DBH).
- **Filter step:** identify a small set of candidate nearest neighbors. In BoostMap, this is done by comparing $F(Q)$ with the embeddings of all database objects (which can be done

orders of magnitude faster than computing the chamfer distance between the input image and the database images), and selecting a small number of database objects whose embeddings are the closest to $F(Q)$. In DBH, we simply select all the objects found in the l bins that the input image hashes to.

- **Refine step:** Compute the exact chamfer distance between Q and each of the database objects selected during the filter step.
- **Output:** return the database object (among all objects considered in the refine step) with the smallest chamfer distance to the input image.

The filter step provides a preliminary set of candidate nearest neighbors in an efficient manner, that avoids computing the exact chamfer distance between the input image and the vast majority of database images. The refine step applies the exact chamfer distance only to those few candidates. Assuming that the mapping step and the filter step take negligible time (a property that is demonstrated in the experiments), filter-and-refine retrieval is much more efficient than brute-force retrieval.

8. EXPERIMENTS

The database of hand images used in the experiments has been constructed as described in Sec. 3. The test set consists of 710 images. All test images were obtained from video sequences of a native ASL signer either performing individual handshapes in isolation or signing in ASL. The hand locations were extracted from those sequences using the method described in [40]. The test images are obtained from the original frames by extracting the subwindow corresponding to the hand region, and then performing the same normalization that we perform for database images, so that the image size is 256×256 pixels, and the minimum enclosing circle of the hand region is centered at pixel $(128, 128)$, and has radius 120. Examples of test images and their corresponding edge images (edge images are used for the chamfer distance computation) are shown in Fig. 3.

For each test image, filter-and-refine retrieval is performed to identify the nearest neighbor of the test image. BoostMap or DBH are used for the filter step. The test image is considered to have been classified correctly if the handshape of the nearest neighbor is the same as the handshape of the test image. The ground truth for the test images is manually provided. The total number of handshapes is 20, so our classification task consists of recognizing 20 distinct classes.

Figures 5 and 6 illustrate the results obtained with BoostMap and DBH on our data set. There are two types of results: results on retrieval accuracy, and results on classification accuracy. Nearest neighbor *retrieval accuracy* is the fraction of test images (out of the 710 images in our test set) for which the retrieved nearest neighbor (using filter-and-refine retrieval) was the true nearest neighbor (according to the chamfer distance) that would have been found using brute-force search. Nearest neighbor *classification accuracy* is the fraction of test images for which the retrieved nearest neighbor is an image of the same handshape as the handshape of the test image.

In terms of retrieval performance, Fig. 5 shows that both BoostMap and DBH achieve remarkable speedups over brute-force search at the cost of some losses in retrieval accuracy (naturally, brute-force search has a retrieval accuracy of 100%). At the same time, we notice that BoostMap performs significantly better than DBH. For example, for 90% retrieval accuracy, BoostMap yields a speedup factor of about 300 over brute-force search, whereas DBH yields a speedup factor of about 26. Similarly, for 99% retrieval accuracy, BoostMap yields a speedup factor of about 59, and DBH yields a speedup factor of about 10.

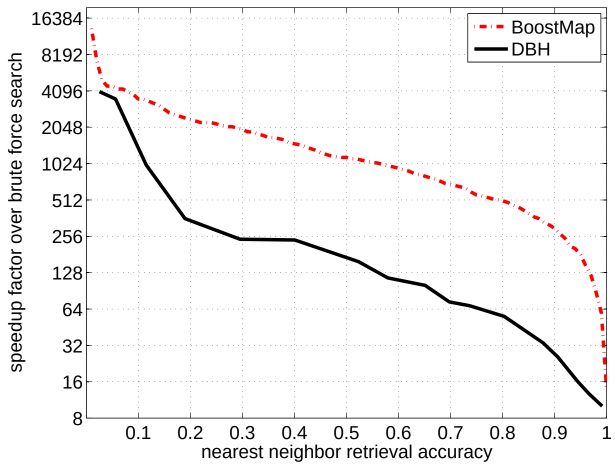

Figure 5: Retrieval speed-up vs. accuracy for BoostMap and DBH. For each accuracy, the plot shows the corresponding speedup factor obtained using BoostMap and DBH. Brute-force nearest neighbor search yields a retrieval accuracy of 1 and an average retrieval time of 112 seconds per query, corresponding to a speedup factor of 1.

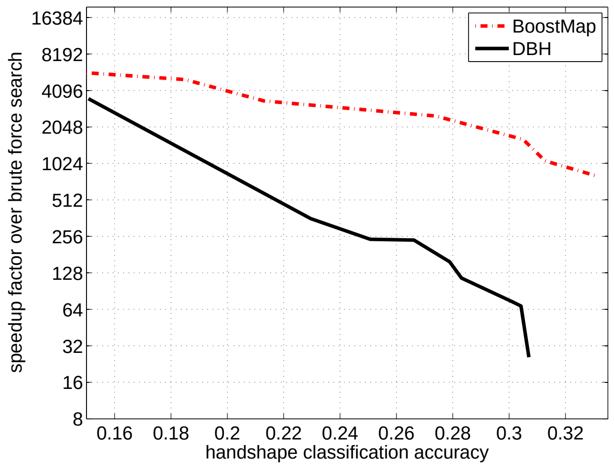

Figure 6: Classification speed-up vs. accuracy for BoostMap and DBH. For each accuracy, the plot shows the corresponding speedup factor obtained using BoostMap and DBH. Brute-force nearest neighbor search yields a classification accuracy of 33.1% and an average retrieval time of 112 seconds per query, corresponding to a speedup factor of 1.

We get a similar picture when we look at classification performance in Fig. 6. An important thing to note here is that the classification accuracy of brute force search (i.e., before we introduce any errors caused by our indexing schemes) is only 33.1%. This accuracy rate reflects the upper limit of how well we can do using our indexing schemes: even if we have an indexing scheme that gives perfect retrieval accuracy with enormous speedups, the classification accuracy is it still going to be the same as that of brute-force search. At the same time, it is important to note that this accuracy rate is obtained without using any domain-specific constraints, and such constraints are oftentimes available, and highly informative, in concrete real-world applications. In Section 9 we discuss some frequently encountered types of constraints, and how they can be used to improve handshape classification accuracy.

With respect to the classification performance obtained using our indexing schemes (BoostMap and DBH), we notice that the speedups that we obtain over brute-force search are impressive, especially for the BoostMap method. With BoostMap, we can get the exact same accuracy rate (33.1%) as with brute-force search, but about 800 times faster. This means that classification time is reduced from 112 seconds per query (using brute-force search) to 0.14 seconds per query. With DBH, we obtain a speedup factor of about 26 for a classification accuracy of 30.7%, which is a slight decrease over the 33.1% accuracy rate of brute-force search.

While the experiments show the need for more research, to design image matching methods that are more accurate that the chamfer distance (some recent progress on that topic is reported at [34]), the experiments also illustrate the power of BoostMap and DBH as indexing methods. BoostMap yields a classification time that is about three orders of magnitude faster than that of brute-force search. While DBH does not perform as well, it also achieves significant speedups with respect to brute-force search.

There is one advantage of DBH over BoostMap that is masked by these experiments. In BoostMap, L_1 distances must be computed between the embedding of the query image and the embeddings of all database images. The overhead of computing these distances takes negligible time in our experiments, but in certain cases (larger datasets, very high-dimensional embeddings, distance measures more efficient than the chamfer distance), this overhead can become significant. DBH does not incur such an overhead.

In summary, the experiments demonstrate that using a large database of hand images is a scalable and feasible approach for recognizing handshapes at arbitrary 3D orientations. Indexing methods can allow database retrieval to operate at interactive speeds. Incorporating informative domain-specific constraints, as discussed in the next section, can bring classification accuracy up to a satisfactory level for specific real-world applications.

9. DISCUSSION AND FUTURE WORK

Our topic in this paper has been robust recognition of handshapes, for the purpose of human computer interaction in real-world applications such as intelligent homes and assistive environments. The same handshape can have very different appearances depending on the 3D orientation of the hand with respect to the camera, and the database-based method described here is well suited to the task of capturing this wide variability in appearance for each handshape.

Recognizing handshapes in arbitrary 3D orientations remains a challenging task, as evidenced by the high error rates in our experiments. However, these error rates correspond, in some sense, to a worst-case scenario, where no prior information is available as to what 3D orientations and handshapes are most likely to be observed. Such prior information is oftentimes available in real-world systems, and can come from the following sources:

- Specific usage scenarios, where the user is typically facing in a certain direction with respect to the camera and makes handshapes with a limited range of 3D orientations.
- Knowledge of specific human-computer communication protocols, that involve a relatively small number of gestures, thus restricting possible handshapes and 3D orientations.
- Use of multiple cameras, which can resolve ambiguities that are unavoidable in systems that only use a single camera.
- Use of linguistic constraints in the context of sign language recognition. For example, given the handshape of the dominant hand there is a relatively small number of possible handshapes for the non-dominant hand.
- Use of information from multiple consecutive frames in a video sequence. The method described in this paper can be a source of hypotheses for initializing a hand tracker. Such a tracker can use information from multiple frames to improve upon the accuracy of estimates made based on a single frame.

As our goal in this paper has been to describe a general-purpose handshape recognition method, incorporating such domain-specific knowledge is beyond the scope of this paper. At the same time, evaluating the method presented here in real-world applications is a very interesting direction for future work. We are particularly interested in integrating our method into a computer vision-based sign language recognition system, for the purpose of designing efficient information access tools for users of sign languages.

10. CONCLUSION

This paper has presented a database-based method for handshape recognition in the context of human computer interaction in real-world applications. We have shown that using a large database of synthetic hand images is a feasible and promising method for capturing the wide range of variability in the appearance of each individual handshape. A key issue that this paper has addressed is the ability of such a method to operate at interactive speeds, given the large number of database images that need to be matched with each input image. We have discussed two nearest neighbor search methods, BoostMap and Distance-Based Hashing, and we have shown that these methods are effective and allow input images to be processed at interactive speeds, with relatively small decreases in recognition accuracy. We believe that integrating this approach with well-grounded domain-specific constraints available for specific applications can lead to efficient and robust handshape recognition in real-world environments.

11. REFERENCES

[1] A. Andoni and P. Indyk. Efficient algorithms for substring near neighbor problem. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1203–1212, 2006.

[2] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 459–468, 2006.

[3] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios. Boostmap: An embedding method for efficient nearest neighbor retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 30(1):89–104, 2008.

[4] V. Athitsos, M. Hadjieleftheriou, G. Kollios, and S. Sclaroff. Query-sensitive embeddings. *ACM Transactions on Database Systems (TODS)*, 32(2), 2007.

[5] V. Athitsos, M. Potamias, P. Papapetrou, and G. Kollios. Nearest neighbor retrieval using distance-based hashing. In *IEEE International Conference on Data Engineering (ICDE)*, 2008.

[6] H. Barrow, J. Tenenbaum, R. Bolles, and H. Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *International Joint Conference on Artificial Intelligence*, pages 659–663, 1977.

[7] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):509–522, 2002.

[8] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, 2001.

[9] J. Bourgain. On Lipschitz embeddings of finite metric spaces in Hilbert space. *Israel Journal of Mathematics*, 52:46–52, 1985.

[10] J. Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17(5), 2001.

[11] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[12] Curious Labs, Santa Cruz, CA. *Poser 5 Reference Manual*, August 2002.

[13] T. E. de Campos and D. W. Murray. Regression-based hand pose estimation from multiple cameras. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 782–789, 2006.

[14] C. Faloutsos and K. I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *ACM International Conference on Management of Data (SIGMOD)*, pages 163–174, 1995.

[15] W. Freeman and M. Roth. Computer vision for computer games. In *Automatic Face and Gesture Recognition*, pages 100–105, 1996.

[16] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *International Conference on Very Large Databases*, pages 518–529, 1999.

[17] K. Grauman and T. J. Darrell. Fast contour matching using approximate earth mover's distance. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages I: 220–227, 2004.

[18] T. Heap and D. Hogg. Towards 3D hand tracking using a deformable model. In *Face and Gesture Recognition*, pages 140–145, 1996.

[19] G. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):530–549, 2003.

[20] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems*, 28(4):517–580, 2003.

[21] G. Hristescu and M. Farach-Colton. Cluster-preserving embedding of proteins. Technical Report 99-50, CS Department, Rutgers University, 1999.

[22] C. Li, E. Chang, H. Garcia-Molina, and G. Wiederhold. Clustering for approximate similarity search in high-dimensional spaces. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):792–808, 2002.

[23] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. In *IEEE Symposium on Foundations of Computer Science*, pages 577–591, 1994.

[24] S. Lu, D. Metaxas, D. Samaras, and J. Oliensis. Using multiple cues for hand tracking and model refinement. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 443–450, 2003.

[25] J. Rehg. *Visual Analysis of High DOF Articulated Objects with Application to Hand Tracking*. PhD thesis, Electrical and Computer Eng., Carnegie Mellon University, 1995.

[26] R. Rosales, V. Athitsos, L. Sigal, and S. Sclaroff. 3D hand pose reconstruction using specialized mappings. In *ICCV*, volume 1, pages 378–385, 2001.

[27] R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[28] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter-sensitive hashing. In *IEEE International Conference on Computer Vision*, pages 750–757, 2003.

[29] N. Shimada, K. Kimura, and Y. Shirai. Real-time 3-D hand posture estimation based on 2-D appearance retrieval using monocular camera. In *Recognition, Analysis and Tracking of Faces and Gestures in Realtime Systems*, pages 23–30, 2001.

[30] B. Stenger, A. Thayananthan, P. H. S. Torr, and R. Cipolla. Model-based hand tracking using a hierarchical bayesian filter. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1372–1384, 2006.

[31] A. Thayananthan, B. Stenger, P. H. S. Torr, and R. Cipolla. Shape context and chamfer matching in cluttered scenes. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 127–133, 2003.

[32] E. Tuncel, H. Ferhatosmanoglu, and K. Rose. VQ-index: An index structure for similarity searching in multimedia databases. In *Proc. of ACM Multimedia*, pages 543–552, 2002.

[33] J. Uhlman. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, 1991.

[34] J. Wang, V. Athitsos, S. Sclaroff, and M. Betke. Detecting objects of variable shape structure with hidden state shape models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 30(3):477–492, 2008.

[35] X. Wang, J. T. L. Wang, K. I. Lin, D. Shasha, B. A. Shapiro, and K. Zhang. An index structure for data mining and clustering. *Knowledge and Information Systems*, 2(2):161–184, 2000.

[36] R. Weber and K. Böhm. Trading quality for time with nearest-neighbor search. In *International Conference on Extending Database Technology: Advances in Database Technology*, pages 21–35, 2000.

[37] Y. Wu and T. Huang. View-independent recognition of hand postures. In *CVPR*, volume 2, pages 88–94, 2000.

[38] Y. Wu, J. Lin, and T. Huang. Capturing natural hand articulation. In *ICCV*, volume 2, pages 426–432, 2001.

[39] P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, 1993.

[40] Q. Yuan, S. Sclaroff, and V. Athitsos. Automatic 2D hand tracking in video sequences. In *IEEE Workshop on Applications of Computer Vision*, pages 250–256, 2005.