Using Self-Organizing Maps to Learn Geometric Hash Functions for Model-Based Object Recognition

George Bebis[1], Michael Georgiopoulos[2], and Niels da Vitoria Lobo[3]

Department of Computer Science, University of Nevada, Reno, NV 89557[1]
Department of Electrical & Computer Engineering, University of Central Florida, Orlando, FL 32816[2]
Department of Computer Science, University of Central Florida, Orlando, FL 32816[3]

E-mail: bebis@cs.unr.edu, mng@ece.engr.ucf.edu, niels@cs.ucf.edu

correspondence should be addressed to

Dr. George Bebis
Department of Computer Science
University of Nevada
Reno, NV 89557
Tel: (702) - 784 - 6463
Fax: (702) - 784 - 1877

Abstract

A major problem associated with geometric hashing and methods which have emerged from it is the non-uniform distribution of invariants over the hash space. This has two serious effects on the performance of the method. First, it can result in an inefficient storage of data which can increase recognition time. Second, given that geometric hashing is highly amenable to parallel implementation, a non-uniform distribution of data poses difficulties in tackling the load-balancing problem. Finding a "good" geometric hash function which redistributes the invariants uniformly over the hash space is not easy. Current approaches make assumptions about the statistical characteristics of the data and then use techniques from probability theory to calculate a transformation that maps the non-uniform distribution of invariants to a uniform one. In this paper, a new approach is proposed based on an *elastic hash table*. In contrast to existing approaches which try to redistribute the invariants over the hash bins, we proceed oppositely by distributing the hash bins over the invariants. The key idea is to associate the hash bins with the output nodes of a Self-Organizing Feature Map (SOFM) neural network which is trained using the invariants as training examples. In this way, the location of a hash bin in the space of invariants is determined by the weight vector of the node associated with the hash bin. During training, the SOFM spreads the hash bins proportionally to the distribution of invariants (i.e., more hash bins are assigned to higher density areas while less hash bins are assigned to lower density areas) and adjusts their size so that they eventually hold almost the same number of invariants. The advantage of the proposed approach is that it is a process that adapts to the invariants through learning. Hence, it makes absolutely no assumptions about the statistical characteristics of the invariants and the geometric hash function is actually computed through learning. Furthermore, SOFM's "topology preserving" property ensures that the computed geometric hash function should be well behaved. The proposed approach, was shown to perform well on both artificial and real data.

1. Introduction

During the last two decades, there has been a variety of approaches to tackle the problem of object recognition. The most successful approach is probably in the context of *model-based* object recognition [1], where the environment is rather constrained and recognition relies upon the existence of a set of predefined object models. The indexing based approach to object recognition has been prevalent for quite a number of years. According to this approach, a model database is built first by establishing proper associations between features and models. Then, during the recognition stage, scene features are used to retrieve the right associations stored in the model database. Efficient indexing schemes are used for both organizing and searching the model database effectively. Geometric hashing [2] is a well known technique which belongs into this category. It is based on the idea of storing information about the models in a table, using a hashing scheme. The indexing of the appropriate hash bin where this information must be stored is performed using transformation invariant object features called *invariants*. During recognition, the same hashing scheme is used in order to retrieve the most feasible models from the model database.

Geometric hashing and approaches which have emerged from it suffer from a major problem: the non-uniform distribution of invariants over the hash space. Taking into consideration that geometric hashing is amenable to parallel implementation [6][7], a uniform distribution of data is highly desirable for solving the load-balancing problem (i.e., distributing the data over the processors) [6]. Also, the non-uniform nature of the distribution of invariants results in an inefficient storage of the data over the hash table which can slow down recognition significantly. The key solution to the problem is the selection of a "good" geometric hash function which can redistribute the data uniformly over the hash table. In addition, it is very important that the hash function is proximity preserving, that is, it it maps similar data to hash bins located close together. Hash functions which preserve proximity are very desirable because partial voting, a heuristic which increases geometric hashing's noise tolerance, can be implemented efficiently (see section 4).

It is well known, however, that "good" hash functions are difficult to find. In [11], several hash functions were considered and evaluated to find which one performs best. The conclusion was that the selection of a good hash function is data dependent. Rehashing is a different approach which has been recently suggested [4][5]. The idea is to compute a transformation which maps the distribution of invariants to a uniform one, using techniques from probability theory. Although rehashing is an interesting approach, it has two drawbacks: first, it is based on the assumption that the probability density function (*pdf*) of the model point

features is known *a priori* (the pdf of model point features is required in the calculation of the distribution of invariants). Second, the derivation of the rehashing transformation involves complex calculations which, in certain cases, are even intractable [5].

In this paper, a new approach is presented which does not make any assumption about the statistical characteristics of the distribution of model points and does not require the estimation of the pdf of invariants. A shorter version of this work was presented in [8]. In geometric hashing, hash bins correspond to locations in the space of invariants. The location of a hash bin is very critical because it determines which invariants will access the hash bin. Common approaches distribute the hash bins in a way that does not always resemble the distribution of invariants. More efficient approaches, like rehashing, distribute the hash bins uniformly but then try to distribute the invariants uniformly over the hash bins. In contrast to these approaches, we proceed oppositely by distributing the hash bins over the invariants, without redistributing the invariants. In other words, the hash table used is not static but rather *elastic*. The key idea is to associate the hash bins with the output nodes of a SOFM neural network which is trained using the invariants as training examples. In this way, the location of a hash bin is determined by the weight vector of the node associated with the hash bin. The behavior of the SOFM during training resembles an *elastic grid* which deforms over the space of invariants. The objective of the deformation process is to distribute the weight vectors (i.e., hash bins' locations) according to the distribution of invariants. The training of the SOFM is performed using a variation of the Kohonen algorithm [9], motivated by [13], which we call the *Kohonen algorithm with conscience*.

The proposed approach has the advantage that the hash function, implemented by the SOFM, is actually computed through learning. Since the choice of a proper hash function seems to be problem dependent, the availability of a scheme which automatically finds a "good" hash function for a given problem, through learning, is highly desirable. Also, the topology preserving property [9] implies that the computed hash function should be well behaved, that is, it should be piece-wise continuous and should not have singularities. Thus, the learned hash function will be proximity preserving. Finally, the proposed approach is notable for its simplicity and it is inherently parallelizable.

The organization of the paper is as follows: Section 2 discusses geometric hashing. Section 3 presents a brief overview of the SOFM. In section 4 we show how the SOFM can be used to solve the problems caused by the non-uniformity of invariants. In section 5, we discuss the problem of node underutilization and we present an experimental study involving a number of variations of the Kohonen algorithm. Implementation

details, experimental results, and comparisons with existing approaches are given in section 6. Finally, section 7 presents our conclusions.

2. Geometric hashing and rehashing

Geometric hashing is based on the idea of storing redundant, transformation-invariant, information about an object in a database. During preprocessing, a number of feature points are extracted and the objects are represented in an affine invariant way. This is performed as follows: first, three non-collinear points (*basis triplet*) are chosen from the set of feature points and a coordinate frame based on these points is defined. Then, the coordinates of all other feature points are recomputed in terms of the new coordinate frame defined. Figure 1 shows the new coordinates (u,v) of point p_i, computed in the coordinate system defined by p_1, p_2 and p_3. The new coordinates are called *invariants* because they remain unchanged to affine transformations of the object, assuming that the same basis triplet is chosen [2]. The same procedure is repeated for all possible triplets which can be formed by changing the order of the points in the triplet or choosing new points from the set of feature points. The recomputed coordinates (u, v) are used, after proper quantization, as an index into a hash table where an entry (composed of the *basis-triplet* and *model*) is recorded.

<Figure 1 - about here>

During the recognition step, the hash table is used to determine which models are present in the scene. First, an arbitrary ordered triplet of non-collinear points is chosen from the scene. Then, the coordinates of the remaining scene points are recomputed in terms of the coordinate frame defined by this triplet. The recomputed coordinates of each point are used as an index into the hash table and for each entry *(basis-triplet, model)* recorded there, a vote is cast. Entries *(basis-triplet, model)* which score a large number of votes imply possible matches between the model triplets they store and the scene triplet chosen. These possible matches (hypotheses) are then verified by seeking further evidence to support them. Figure 2 demonstrates the preprocessing and recognition steps.

<Figure 2 - about here>

The efficiency of the geometric hashing technique relies heavily on the distribution of invariants over the hash space. The number of hypotheses generated depends on the distribution of invariants. In the extreme case where all the invariants hash into the same hash bin, geometric hashing will be very inefficient since all

possible matches will have to be considered. In general, the invariants are heavily non-uniformly distributed which implies that the hash entries will also be stored non-uniformly over the hash table. Rehashing [4][5] has been proposed as an effective approach for dealing with the non-uniformity of invariants. Specifically, rehashing is a transformation which maps the distribution of invariants to a uniform distribution. This is performed by assuming that the model points are generated by either a Gaussian random process or a process that is uniform over the unit disc or the unit square. Based on these assumptions, the distribution of invariants was calculated.

Three classes of transformations were considered: rigid, similarity, and affine. The transformation which maps point features to invariants can be found easily for each of these cases. Once the pdf of invariants was known for a given geometric transformation, another transformation that maps the pdf of invariants to a uniform distribution was calculated using probability theory techniques again. However, analytical formulas could not be derived in every case because of the intractability of the computations involved [5]. In particular, analytical expressions were derived for all three cases of transformations, only under the assumption that the pdf of model point features was Gaussian with zero mean. In these cases, the distributions of invariants were shown to be variations of the Cauchy distribution with zero mean.

Although rehashing is a mathematically sound approach, it has two drawbacks: first, it is based on the assumption that the model point features are drawn from a known distribution. However, this assumption is not always valid, especially when the number of models is not very large or there is not much variance in the database. Section 6 provides a number of examples. Second, the steps involved in the derivation of the rehashing transformation are complex.

3. The Self Organizing Feature Map

In this section, we present a brief overview of the SOFM and its properties. The SOFM consists of an input layer and a single output layer of nodes which usually form a two-dimensional array. The training of the SOFM is usually performed using the Kohonen algorithm [9]. The are two phases of operation: the similarity matching phase and the weight adaptation phase. Initially, the weights are set to small random values and a pattern is presented to the input nodes of the network. During the similarity matching phase, the distances d_i between the inputs and the weights are computed:

$$d_i = \sum_j (x_j^{\mu} - w_{ij}(t))^2$$

where x^{μ} is the μ-th training pattern and w_{ij}(t) is the weight from input node j to output node i at step t. Next, the output node i^* having the minimum distance d_{i^*} is chosen and is declared as the "winner" node. In the weight adaptation phase, the weights from the inputs to the "winner" node are adapted. In addition, a topological neighborhood $N(i, i^*)$ of the winning node i^* is defined and the weights connecting the inputs to the nodes contained in this topological neighborhood are also adapted. The weight changes are based on the following rule :

$$w_{ij}(t+1) = w_{ij}(t) + \eta(x_j^{\mu} - w_{ij}(t))$$

for $i \in N(i, i^*)$. The parameter η is the learning rate of the algorithm. Generally, the neighborhood $N(i, i^*)$ and the learning rate η are decreasing functions of time [9]. A typical choice for $N(i, i^*)$ is

$$N(i, i^*) = \exp \frac{-|r_i - r_{i^*}|^2}{2\sigma^2(t)}$$

which is equal to 1 for $i = i^*$ and decreases with the distance $(r_i - r_{i^*})$ between units i and i^* in the two-dimensional array (r_i is the location of the node i in the array and $\sigma(t)$ is a width parameter that is gradually decreased) [15]. A common choice for $\eta(t)$ and $\sigma(t)$ is

$$\eta(t) = \eta_0 b^{-\gamma t}$$

$$\sigma(t) = \sigma_0 c^{-\gamma t}$$

where $\gamma = 1/t_{\max}$, and η_0, σ_0, b, c are constants [15]. The training procedure is repeated for a number of steps $t_{\max}$ which is specified *a priori*.

The SOFM possesses some very useful properties. Kohonen [9] has argued that the density of the weight vectors assigned to an input region approximates the density of the inputs occupying this region. In fact, the weight vectors converge to cluster centroids or probability extrema [10]. In other words, after training has been completed, the map will reflect the statistical characteristics of the inputs. Second, the weight vectors tend to be ordered according to their mutual similarity (*topology preserving property*). This property is a direct consequence of the use of topological neighborhoods during training. The importance of this property is that at the end of learning, nearby nodes will respond to similar inputs. This implies that the mapping from the input space to the space of nodes should be well behaved.

4. Alleviating the non-uniformity of invariants using the SOFM

In order to demonstrate the suitability of the SOFM for alleviating the non-uniformity of invariants, we need to examine carefully the hash function employed by geometric hashing. It is a very simple hash function which consists of a linear scaling of the invariants followed by quantization to yield an integer index that fits the dimensions of the hash table. In this case, hashing merely implies a quantization of the space of invariants. Assuming that R^k denotes the space of invariants, hashing defines a partition $D_1, D_2, \ldots, D_N$ of R^k such that:

$$R^k = D_1 \bigcup D_2 \bigcup \cdots \bigcup D_N.$$

Each group D_i is associated with a hash bin, that is, if x belongs to D_i, then hashing will assign x to the hash bin associated with D_i. In order for the hash entries to be distributed uniformly over the hash table, hashing should be able to divide the space of invariants into equiprobable regions.

A simple way to form these regions is by splitting the space of invariants into equal size squared regions. This can be done by projecting a grid of fixed cell size onto the space of invariants. Then if we associate each cell to a hash bin, the invariants falling into the same cell will all be hashed into the hash bin associated with the cell. Partitioning the space of invariants in this way will yield good results only if the distribution of invariants is uniform. In this case, each cell will contain almost the same number of invariants (see Figure 3(a)). Obviously, a grid with a fixed cell size will not yield good results when the distribution of invariants is non-uniform. In this case, certain cells will become over-populated while other cells will remain almost empty. One way to deal with this problem is by choosing a variable cell size. For example, assuming that the distribution of invariants resembles a Gaussian distribution, it is reasonable to make the cell size proportional to the distance of the cell from the center of the distribution. Hence, cells close to the center of the distribution will be given small sizes while cells far away will be given large sizes as is illustrated in Figure 3(b). This approach was followed in [2]. A variable cell size will work well as far as the invariants are distributed uniformly around the center of the distribution. However, the shape of the distribution of invariants varies from case to case and in fact, it depends on the number of objects in the database and their geometrical characteristics. Figure 3(c) shows an example of a possible distribution.

<Figure 3 - about here>

In this paper, a new approach is proposed based on an *elastic grid*. According to this approach, the cell

size as well as the locations of the cells are not chosen in an *a-priori* manner, based on assumptions that might not be true, but through a learning procedure which extracts the statistical characteristics of the distribution of invariants and determines the size and location of the cells adaptively. The idea is to move the cells of the grid over the populated regions of the space of invariants and adjust their size in a way that eventually every cell encloses almost the same number of invariants. That is, the cells should be distributed according to the density of the invariants in a particular region.

The key idea in implementing the above approach is to associate the cells of the grid (i.e., hash bins) with the output nodes of a SOFM, which is trained using the invariants as training examples. In this way, the location of a hash bin in the space of invariants is determined by the weight vector of the node associated with the hash bin. During training, sample invariants are presented to the network and the weight vectors change positions according to Kohonen's learning rule given is section 3. The learning procedure distributes the weight vectors (i.e., hash bins) according to the distribution of invariants. After training, the SOFM implements a non-linear mapping from the input space (i.e., space of invariants) to the space of nodes. This mapping actually quantizes the space of invariants by partitioning it into a number of regions D_i. To understand this better, one can visualize the nodes of the network as points in the space of invariants. In general, nodes can be visualized as points in two different spaces: the space of nodes and the input space. A node's location in the space of nodes is just the physical location of the node in the two-dimensional grid of nodes, while a node's location in the input space is the location determined by the weight vector associated with this node. After the locations of the nodes have been plotted in the space of invariants, two points are connected if their corresponding nodes are physical neighbors (i.e., neighbors in the space of nodes) [9]. Figure 4 shows an example in the case of node i and its neighbors. Each node's location has been plotted in the input space and i's location has been connected with the locations of all the other nodes since they are physical neighbors of i in the space of nodes. The region D_i defined by node i contains all the points which are closer to the vector w_i than to any other vector w_j, $i \neq j$.

By representing the nodes of the SOFM as points in the space of invariants and connecting them together according to the procedure described above, the nodes of the SOFM form an *elastic grid* over the space of invariants. During training, the locations of the nodes change since the weights of the network change. Thus, training can be seen as a deformation of an elastic grid over the space of invariants. The objective of the deformation process is to distribute the nodes of the elastic grid proportionally to the distribution of invariants, that is, to assign more nodes in the more crowded areas and less nodes in the lower density

areas. It should be mentioned that by changing the locations of the hash bins, their sizes change as well. This is because the size of a hash bin depends on the locations of its neighboring hash bins as Figure 4 illustrates. The importance of the proposed approach is that no assumptions about the distribution of invariants need to be made and that it is a process that adapts to the invariants.

<Figure 4 - about here>

The SOFM possesses two properties which make it very suitable for the problem at hand. First, the density of the weight vectors approximates the density of the inputs and second, the mapping implemented by the SOFM preserves the topology of the map (see section 3). The first property implies that the space of invariants should be partioned into a number of equiprobable regions. Ideally, the probability of a randomly selected invariant (selected according to the *pdf* of invariants), being closest to any given weigh vector should be $1/m$ where m is the number of output nodes. As a result, each hash bin is expected to hold almost the same number of entries. Unfortunately, this is not quite true in practice and certain heuristics must be incorporated in the the training algorithm to improve results (see next section). The second property implies that the mapping from the space of invariants to the space of "nodes" or "hash bins", will be proximity preserving. Thus, similar invariants will be mapped to hash bins located close together. This property is very desirable for implementing partial voting efficiently. Very briefly, partial voting is a heuristic for improving the noise tolerance of geometric hashing. When noisy data are used to retrieve data from the model data base, it is quite unlikely that the correct hash bins will be accessed. In partial voting, instead of voting for a single hash bin, multiple votes are cast for hash bins in a neighborhood around the targeted hash bin. Although this heuristic increases the number of hypotheses during recognition, it has be shown to be very beneficial and its use is imperative [2][5].

5. Adding conscience to the Kohonen learning algorithm

A serious problem with competitive learning algorithms is that they often lead to solutions where several nodes of the network remain underutilized or completely unutilized. For example, if some region of the input space is more crowded than others and the initial density of weight vectors is too low in this region, specific nodes will be winning the competitions consistently. The Kohonen learning algorithm attempts to overcome this problem by using topological neighborhoods. Although this approach is very effective, it does not alleviate the problems completely. There are a number of approaches in the literature which try to deal with these problem [9]. Three of them, which are quite representative, have been considered here.

The first approach is based on a *convex combination* of the inputs [12]. According to this method, all the initial weights are set to the same value $1/\sqrt{n}$, where n is the dimensionality of the input vectors. Then, each component x_i of the input vector is substituted by the value $\alpha x_i + (1-\alpha)/\sqrt{n}$, where α is turned up gradually from 0 to 1 giving the opportunity to the input vectors to attract the weight vectors slowly. The second approach is called *competitive learning with conscience* [13]. The idea is to associate a threshold with each node of the network. If a node has won more than $1/m$ of the time (where m is the number of nodes in the network), its chance of winning again is reduced by raising its corresponding threshold. The third method is called *competitive learning with attention* [14]. This approach is very similar to [13]. In particular, a counter is assigned to each node of the network which keeps track of the number of times the node wins the competition. Weight vectors are adjusted according to the frequency at which nodes have won in the past. Each of the above methods adapts the weights of the winning node only and the learning rate is assumed to be constant in [13].

<Figure 5 - about here>

TABLE 1. Standard deviation of the votes received by the nodes of the networks.

Standard deviation			
	Uniform	Gaussian	Circle
Convex	3.93	10.49	43.57
Attention	0.77	1.63	3.89
Conscience	0.72	1.0	2.18
Kohonen	3.68	7.30	10.90

All of the above methods were tested using the data sets shown in Figure 5. The number of training epochs was chosen to be 6,000 for the convex combination approach and 1,000 for the other approaches. To estimate the utilization of nodes after training, we presented to the networks all the training patterns once (without changing the weights) and we recorded how many times each node became a winner. We call this procedure "voting" (each input votes for a node), relating it implicitly to the voting scheme of geometric hashing (each invariant votes for a hash bin). Then, we computed the standard deviation of the votes received by each node. The standard deviation is a measure of the utilization of nodes since a small standard deviation indicates that all the nodes are utilized (i.e., every node receives almost the same number of votes) whereas a large standard deviation implies that certain nodes have been underutilized and others have been overutilized. Table 1 shows the standard deviation of the votes received by the nodes of the networks. For comparison purposes, we also included results using the Kohonen algorithm. Clearly, the competitive learning with

conscience approach performed best in all the cases.

<Figures 6,7 - about here>

Figure 6 shows the maps obtained by each approach. As it was expected, only the Kohonen algorithm preserves the topology. This is a direct consequence of the fact that all other algorithms change the weights of the winning node only. However, Kohonen algorithm's performance as far as node utilization is concerned is not satisfactorily. Since the topology preserving property is very critical in the computation of a well behaved hash function, we decided to combine the Kohonen algorithm, which preserves the topology, with each one of the other three approaches, which improve node utilization. Our objective was to strengthen the performance of the Kohonen algorithm in terms of node utilization, while preserving the topology at the same time. Three variations were created in this way for comparison: the Kohonen algorithm using convex combination of inputs (SOFM-CV), the Kohonen algorithm with attention (SOFM-A), and the Kohonen algorithm with conscience (SOFM-C).

Since it was not evident from the beginning whether the incorporation of these heuristics into the Kohonen algorithm will affect the topology forming process or not, all the variations were consistently tested using the same data sets and the same number of training epochs. The results obtained show that all the variations were able to find solutions which preserve the topology. A slight distortion in the computed maps might occur, but it does not seem to be very significant. Among the three variations, the SOFM-C gave the best results in terms of node utilization. Table (2) shows the standard deviation of the votes for each case. Clearly, the node utilization exhibited by the Kohonen algorithm with conscience is superior to the node utilization exhibited by the Kohonen algorithm itself as can be seen by comparing the last rows of Tables (1) and (2). Figure 7 shows the maps found by the SOFM-C. Figure 8 shows the steps involved in the SOFM-C. *C* and *D* are constants associated with the biasing term added to the distance measure of the algorithm.

<Figure 8 - about here>

TABLE 2. Standard deviation of the votes received by the nodes of the networks.

Standard deviation			
	Uniform	Gaussian	Circle
SOFM-CV	3.79	6.52	11.73
SOFM-A	0.88	1.61	3.85
SOFM-C	0.60	0.92	1.38

6. Experimental results

In this section, we present a number of experimental results which demonstrate the effectiveness of the proposed approach. Comparisons with rehashing are also provided.

6.1. Experiment 1

The purpose of this experiment is to examine the performance of rehashing. Similarity transformations have been considered in this experiment. First, we considered the objects shown in Figure 9(a). The distribution of invariants for this data set is shown in Figure 9(c). Applying the rehashing transformation derived for the case of similarity transformations yields the distribution shown in Figure 9(e). Obviously, rehashing performs quite satisfactorily in this case. Next, we considered the objects shown in Figure 9(b). The distribution of invariants computed for this data set is shown in Figure 9(d) while the rehashed distribution is shown in Figure 9(f). Obviously, rehashing does not perform well in this case. According to the theoretical in [4], the rehashing transformation computed for the case of similarity transformations was based on distribution of invariants very similar to the Cauchy distribution. Although the distribution of Figure 9(c) resembles the Cauchy distribution, the distribution of Figure 9(d) is very different. This is the reason rehashing does not perform well in this case. Figures 10(a), and 10(c) show the distribution of hash entries over a 20 x 20 hash table in the case of the original and rehashed invariants correspondingly.

<Figure 9,10,11 - about here>

Next, we trained a SOFM-C with two inputs and 400 output nodes, arranged on a 20 x 20 grid. The network was trained for 500 epochs. The number of invariants used during training was 31,572 and they were normalized in the range [0, 1] x [0, 1]. The initial weights of the SOFM were also chosen from the same range. The parameters of the network were chosen as follows: η_0=1.0, σ_0=20 (equal to the maximum of the dimensions of the feature map), $b = c = 1.0/15t_{\max}$, C=1.0 and D=0.01. The feature map to which the network converged is shown in Figure 11(a). The structure of the feature map illustrates the way the hash bins were distributed over the invariants. Figure 10(e) shows the distribution of entries over the hash table. To estimate the hash table utilization, we computed the standard deviation (SD) of the number of entries stored at each hash bin. The first row of Table (3) shows the results. Obviously, the SOFM-C has found a superior solution. At the beginning of the training process, the SD was 422. Then, it gradually decreased during training as shown in Figure 12(a).

TABLE 3. Standard deviation of the number of hash entries.

Standard deviation			
	Unrehashed	Rehashed	SOFM-C
Similarity transf.	62.31	35.33	13.40
Affine transf.	194.20	86.06	25.38

6.2. Experiment 2

In this experiment, we consider the case of affine transformations. Figure 13 shows the set of the real objects used. Invariants based on unstable basis triplets were rejected using the area based criterion [11]. Figure 14(a) shows the distribution of invariants in this case. The application of the rehashing transformation, derived under the assumption of affine transformations, does not yield good results, as is demonstrated in Figure 14(b). To explain why, let us observe that the third quadrant of the distribution of invariants in Figure 14(a) is less crowded than any the other quadrant. This is in agreement with the qualitative results of [4][5], under the assumption that the distribution of model points is uniform over a convex domain. However, no rehashing transformation was derived under this assumption because of the intractability of the computations involved. The only rehashing transformation derived in the case of affine transformations is based on the assumption that the distribution of model features is Gaussian over a convex domain and this is the one used here. We believe that this is the reason rehashing did not perform well in this example.

<Figures 12,13,14 - about here>

Figures 10(b) and 10(d) show the distribution of hash entries over a 20 x 20 hash table, for the case of the original invariants and rehashed invariants correspondingly. A SOFM-C with the same architecture as in the previous experiment was utilized in order to demonstrate the performance of our approach. The same network parameters were chosen as before except for the number of epochs which was chosen to be 100. The number of invariants used to train the SOFM-C was 41,292, normalized in the range [0, 1] x [0, 1]. The initial weights were chosen from the same range. The map to which the network converged is shown in Figure 11(b). Figure 10(f) illustrates the distribution of hash entries over the hash table while the second row of Table (3) shows the computed SDs (for the hash table utilization). It can be be noticed from Figure 10(f) that several hash bins are still over-populated, especially in the boundaries of the hash table, but most hash bins hold almost the same number of entries. Figure 12(b) shows the decreasing behavior of the SD during training (initial value was 224).

6.3. Experiment 3

One of the goals in redistributing the data over the hash table is to reduce the number of hypotheses during recognition. To illustrate this we have performed a number of recognition experiments. Here, we report two of them. In the first experiment, we considered the scene shown in Figure 15(a). A Laplacian edge detector separated the objects from the background and a boundary following routine extracted their boundaries (see Figure 15(c)). The interest points shown correspond to curvature maxima and zero-crossings of the boundary (22 interest points were extracted) [16]. The recognition results are shown in Figure 15(e) (the correctly recognized models have been back-projected on the scene).

<Figure 15 - about here>

TABLE 4. Number of hypotheses tried during verification.

Number of hypotheses			
	Unrehashed	Rehashed	SOFM-C
Scene1 (Model3)	140998	99329	69835
Scene1 (Model4)	11335	8550	6861
Scene2 (Model1)	320512	292046	204997
Scene2 (Model2)	220435	178334	144546
Scene2 (Model3)	34295	26947	19784

The first two rows of Table (4) show the number of hypotheses tried by each approach until both models recognized correctly (60% or more of the model points were required to match with the scene). As can be observed, the proposed approach verified fewer hypotheses. Next, we considered the scene of Figure 15(b). This is a fairly complicated scene. The same procedure, as above, was applied in order to extract the object boundaries (only the outer boundaries were used in our experiment) and the interest shown in Figure 15(c) (45 interest points were extracted). The recognition results are shown in Figure 15(f) and the number of hypotheses verified by each approach is shown in the last three rows of Table (4). Clearly, the proposed approach has verified fewer hypotheses. Overall, the proposed approach verified about 35% - 50% less hypotheses than the hypotheses verified by geometric hashing without rehashing and 20% - 30% less hypotheses than the hypotheses verified by geometric hashing with rehashing.

7. Discussion and conclusions

In this paper, we considered the geometric hashing technique, an indexing based object recognition method which suffers from the problem of the non-uniform distribution of the data over the hash table. A new approach for alleviating this problem was presented based on the SOFM. The proposed approach has a

number of advantages. First, it is not based on any assumption about the characteristics of the distribution of invariants. Second, the hash function is implemented by the SOFM and is actually computed through learning. Third, the topology preserving property of the SOFM guarantees that the computed hash function should be well behaved. The availability of a learning scheme which can be used to find a geometric hash function having nice properties, independently of the problem at hand, is particularly attractive. The independence of the proposed approach from any assumption and the good behavior of the solutions obtained suggest that it might be a useful tool in helping us to derive approximate analytical rehashing functions in cases where a closed form solution cannot be found using traditional approaches.

One disadvantage of the proposed approach is that the solutions obtained are sensitive to the selection of certain parameter values, namely, the number of training epochs $t_{\max}$ and the parameter C used in the modified distance measure of the SOFM-C. Both parameters were chosen by trial and error during our experimentation. We believe that further improvements in the solutions found by the SOFM-C are possible (i.e., solutions with lower SDs). However, this requires extensive experimentation. It is should be mentioned that after the completion of our work, a new, improved version of the competitive learning with conscience approach came to our attention [17]. Specifically, it was shown that the choice of the parameter C is data dependent and a new algorithm which chnages C adaptively during learning was introduced. We strongly believe that this approach can further improve our results (i.e., obtain smaller SDs).

Acknowledgements

The first author is particularly obliged to Dr. Isidore Rigoutsos for his valuable comments and help. This work was supported by a grant from the FSGC (Florida Space Grant Consortium) and TRDA (Technological Research and Development Authority).

References

[1] R. Chin and C. Dyer, "Model-based recognition in robot vision", *Computing Surveys*, vol. 18, no. 1, pp. 67-108, 1986.

[2] Y. Lamdan, J. Schwartz and H. Wolfson, "Affine invariant model-based object recognition", *IEEE Trans. on Robotics and Automation*, vol. 6, no. 5, pp. 578-589, October 1990.

[3] Y. Lamdan, J. Schwartz, & H. Wolfson, "On recognition of 3D objects from 2D images", *IEEE*

International Conference on Robotics and Automation, pp. 1407-1413, Philadelphia, April, 1988.

[4] I. Rigoutsos and R. Hummel, "Several results on affine invariant geometric hashing", *In Proceedings of the 8th Israeli Conference on Artificial Intelligence and Computer Vision*, Tel Aviv, Israel, December 1991.

[5] I. Rigoutsos, *Massively parallel bayesian object recognition*, Ph.D. dissertation, Computer Science Department and Courant Institute of Mathematical Sciences, New York University, 1992.

[6] I. Rigoutsos and R. Hummel, "Massively parallel model matching: geometric hashing on the connection machine", *IEEE Computer*, pp. 33-42, February 1992.

[7] O. Bourdon and G. Medioni "Object recognition using geometric hashing on the connection machine", *International Conference on Pattern Recognition (ICPR)*, Vol II, pp. 596-600, Atlantic City, New Jersey, 1990.

[8] G. Bebis, M. Georgiopoulos and N. da Vitoria Lobo, "Learning Geometric Hashing Functions for Model Based Object Recognition", *Fifth International Conference on Computer Vision (ICCV-95)*, pp. 543-548, Boston, Massachusetts, June 1995.

[9] T. Kohonen, *Self-organizing maps*, Springer-Verlag, 1995.

[10] B. Kosko, "Stochastic competitive learning", *IEEE Transactions on Neural Networks*, vol. 2, no. 5, pp. 522-529, 1991.

[11] M. Costa, R. Haralick, and L. Shapiro, "Optimal affine matching", *In Proceedings of the 6th Israeli Conference on Artificial Intelligence and Computer Vision*, Tel Aviv, Israel, December 1989.

[12] R. Hecht-Nielsen, "Counterpropagation networks", *Applied Optics*, vol. 26, pp. 4979-4984, 1987.

[13] D. DeSieno, "Adding a conscience to competitive learning", *IEEE International Conference on Neural Networks*, vol. I, pp. 117-124, San Diego, California, 1988.

[14] G. Huerter, "Solution of the traveling salesman problem with an adaptive ring", *IEEE International Conference on Neural Networks*, vol. I, pp. 85-92, San Diego, California, 1988.

[15] H. Ritter and K. Schulten, "Convergence properties of Kohonen's topology preserving maps: fluctuations, stability, and dimension selection", *Biological Cybernetics*, vol. 60, pp. 59-71, 1988.

[16] F. Mokhtarian and A. Mackworth, "A theory of multiscale, curvature-based shape representation for planar curves", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 8, pp.

789-805, 1992.

[17] L. Chen and S. Chang, "An adaptive conscientious competitive learning algorithm and its applications", *Pattern Recognition*, vol. 27, no. 12, pp. 1787-1813, 1994.

Figure Captions

Figure 1. An illustration of the computation of invariants.

Figure 2. A demonstration of the geometric hashing algorithm. In the preprocessing step, a model basis triplet is chosen and the coordinates (invariants) of each other point are computed in the coordinate system defined by this triplet. Then, the hash table is accessed using the coordinates as indexes. In the recognition step, a scene triplet is chosen and the coordinates of all other scene points are computed in the coordinate frame defined by the scene triplet. Then, the hash table is accessed again using the coordinates as indexes. If the scene triplet chosen during the recognition step corresponds to a model triplet chosen during the preprocessing step, then the same hash bins will be accessed. This is illustrated in the example where the scene triplet p_1', p_2', and p_3' corresponds to the model triplet p_1, p_2, and p_3.

Figure 3. Partioning the space of invariants into regions.

Figure 4. An illustration of the regions formed by the SOFM.

Figure 5. The data sets.

Figure 6. The maps obtained using the convex combination (first row), competitive learning with attention (second row), competitive learning with conscience (third row) and Kohonen (last row) algorithms.

Figure 7. The maps produced using the SOFM-C

Figure 8. The Kohonen algorithm with conscience (SOFM-C).

Figure 9. (a) The set of numbers, (b) the set of knives, (c) the distribution of invariants for the set of numbers, (d) the distribution of invariants for the set of knives, (e) the rehashed distribution of invariants for the set of numbers, (f) the rehashed distribution of invariants for the set of knives.

Figure 10. (a), (c), (e) the distribution of hash entries under similarity transformations (Figure 9d) using the original approach, rehashing, and the SOFM-C correspondingly, (b), (d) (f) the distribution of hash entries under affine transformations (Figure 14a) using the original approach, rehashing, and the SOFM-C correspondingly.

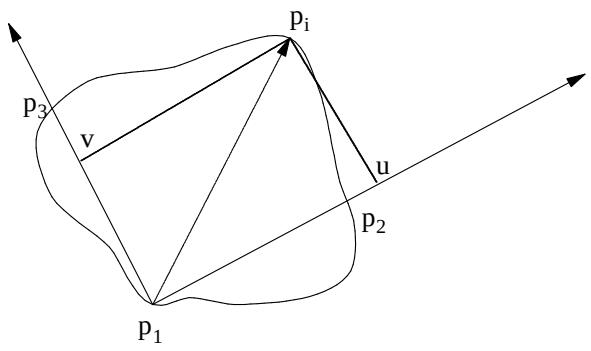
Figure 11. the structure of the SOFM-C for the case of (a) similarity transformations (Figure 9d), and (b) affine transformations (Figure 14a).

Figure 12. The improvement of the standard deviation during training in the case of (a) the objects shown in Figure 9(b) (assuming similarity transformations) and (b) the objects shown in Figure 13 (assuming affine transformations).

Figure 13. The set of different models.

Figure 14. (a) The distribution of (affine) invariants for the set of different objects, (b) the rehashed distribution of invariants.

Figure 15. (a) and (b) two real scenes with overlapped models, (c) and (d) the boundary contours with the interest points (curvature maxima and zero-crossings) marked, (e) and (f) the recognition results.



$$p_i - p_1 = u(p_2 - p_1) + v(p_3 - p_1)$$

Figure 1

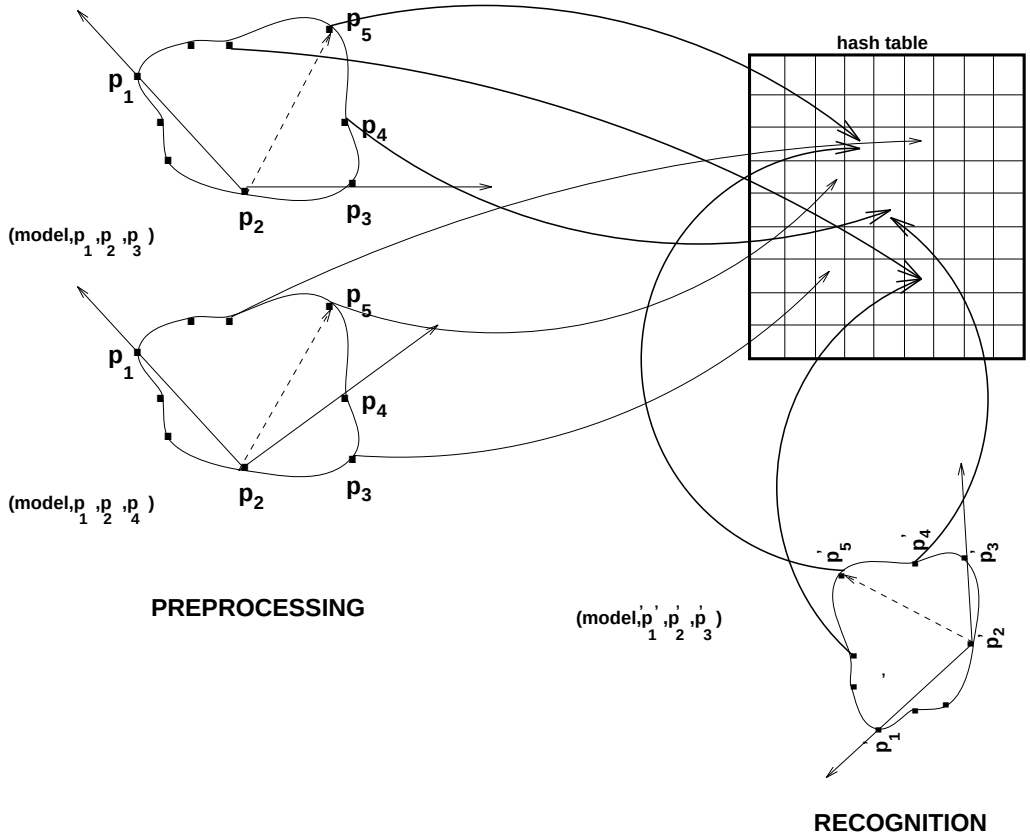
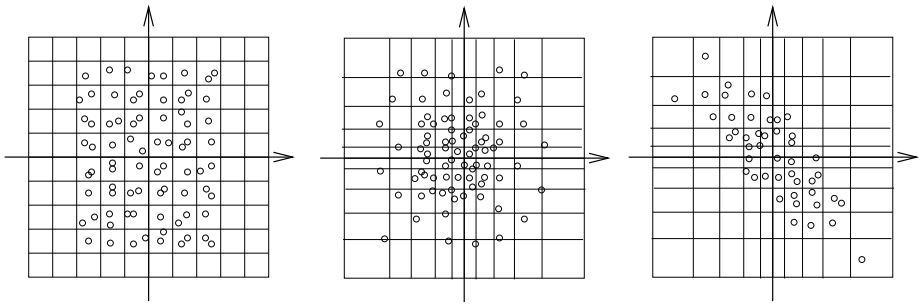

hash table
p5
p1
p4
p2
p3
(model,p1 ,p2 ,p3)
p5
p1
p4
p2
p3
(model,p1 ,p2 ,p4)
PREPROCESSING
p'5
p'4
p'3
p'2
p'1
(model,p'1 ,p'2 ,p'3)
RECOGNITION

Figure 2



(a) (b) (c)

Figure 3

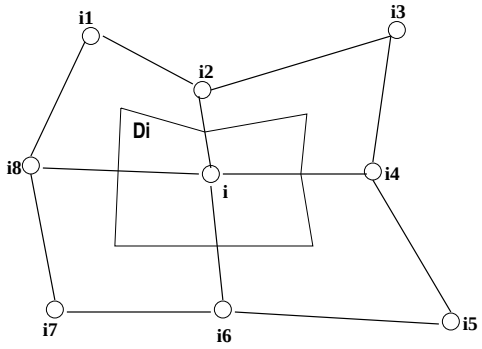


Figure 4

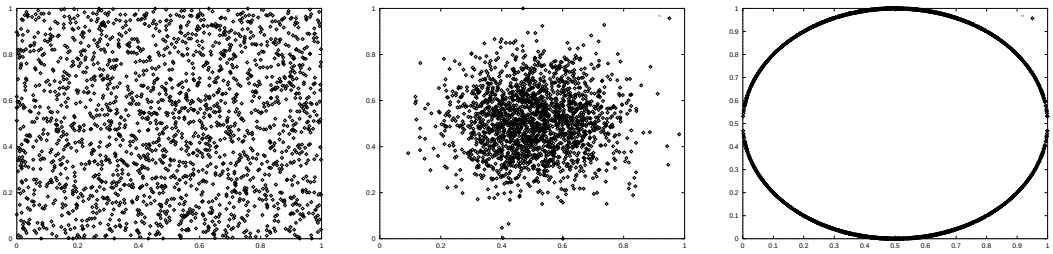


Figure 5

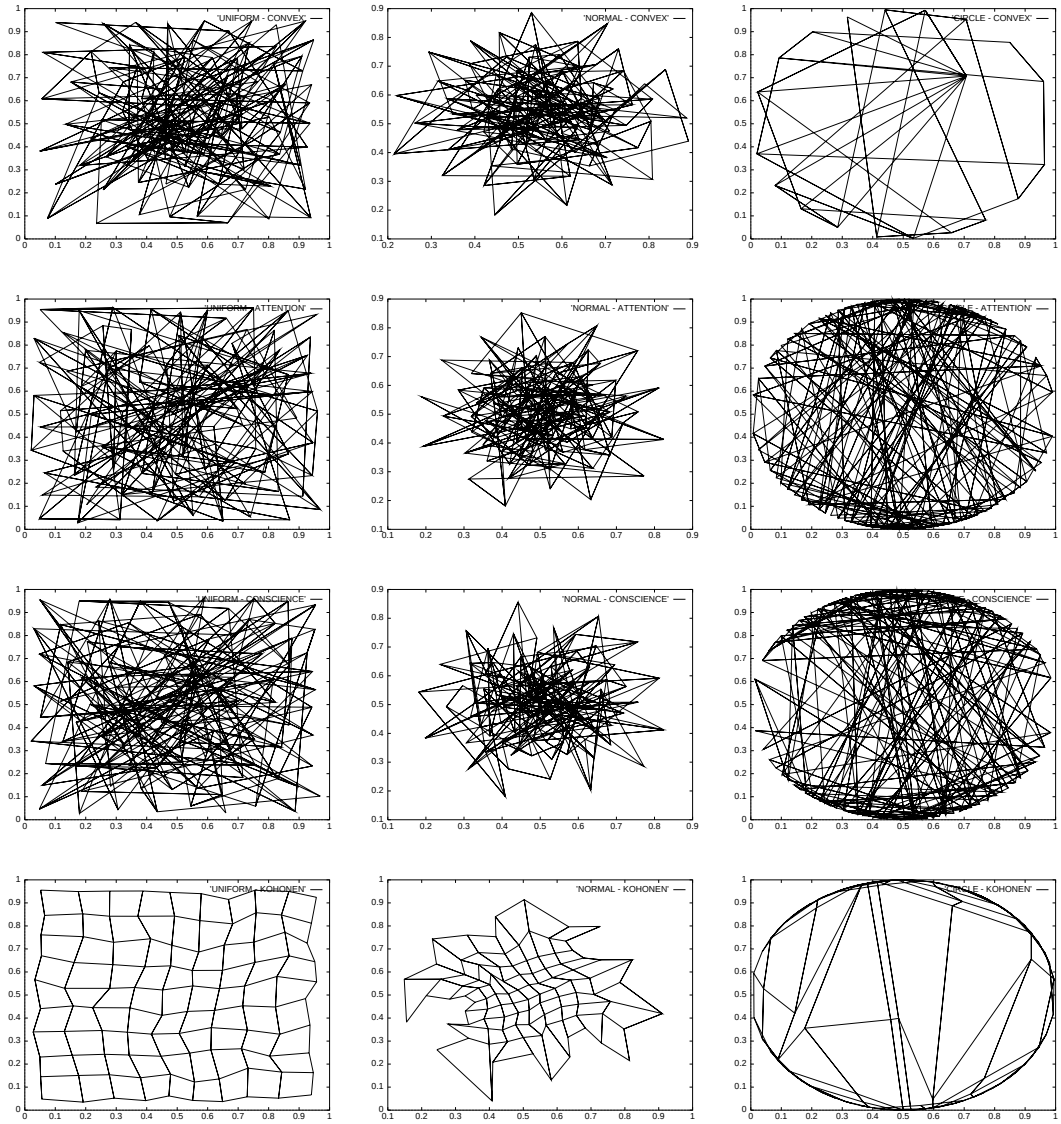

'UNIFORM - CONVEX'
'NORMAL - CONVEX'
'CIRCLE - CONVEX'
'UNIFORM - ATTENTION'
'NORMAL - ATTENTION'
'CIRCLE - ATTENTION'
'UNIFORM - CONSCIENCE'
'NORMAL - CONSCIENCE'
'CIRCLE - CONSCIENCE'
'UNIFORM - KOHONEN'
'NORMAL - KOHONEN'
'CIRCLE - KOHONEN'

Figure 6

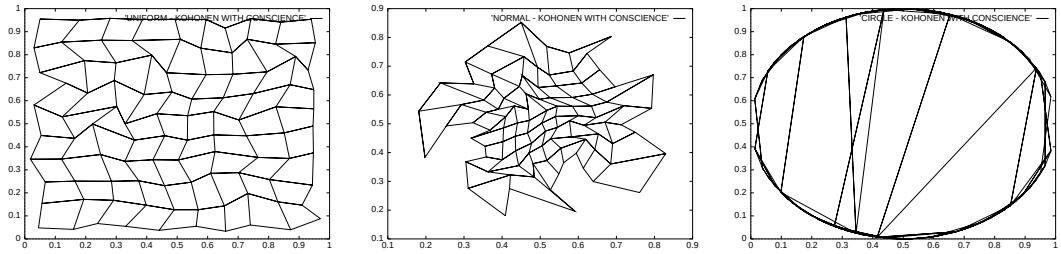

'UNIFORM - KOHONEN WITH CONSCIENCE'
'NORMAL - KOHONEN WITH CONSCIENCE'
'CIRCLE - KOHONEN WITH CONSCIENCE'

Figure 7

1. Initialize the network

Define w_{ij} $(1 \le j \le n, 1 \le i \le m)$ to be the weight from input j to node i, where n is the dimensionality of inputs and m is the number of nodes of the network. Set the initial weights to small random values. Set the biases $p_i(0)$ equal to $1/m$. Choose the number of training steps $t_{\max}$.

2. Similarity matching phase

Present an input pattern and compute the biased distance between the input and each weight vector associated with the output nodes

$$d_i = \sum_j (x_j^\mu - w_{ij}(t))^2 - C\left(\frac{1}{m} - p_i(t)\right)$$

3. Select the minimum distance d_{i^*}

$$d_{i^*} = \min_i d_i$$

4. Update the bias associated with the winning node

$$p_{i^*}(t+1) = p_{i^*}(t) + D(1 - p_{i^*}(t))$$

5. Weight adaptation phase

Update weights for node i^* and nodes contained in the neighborhood $N(i, i^*)$ of the winning node

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t) N(i, i^*)(x_j^\mu - w_{ij}(t))$$

6. if $t = t_{\max}$, then stop; otherwise repeat by going to step 2.

Figure 8

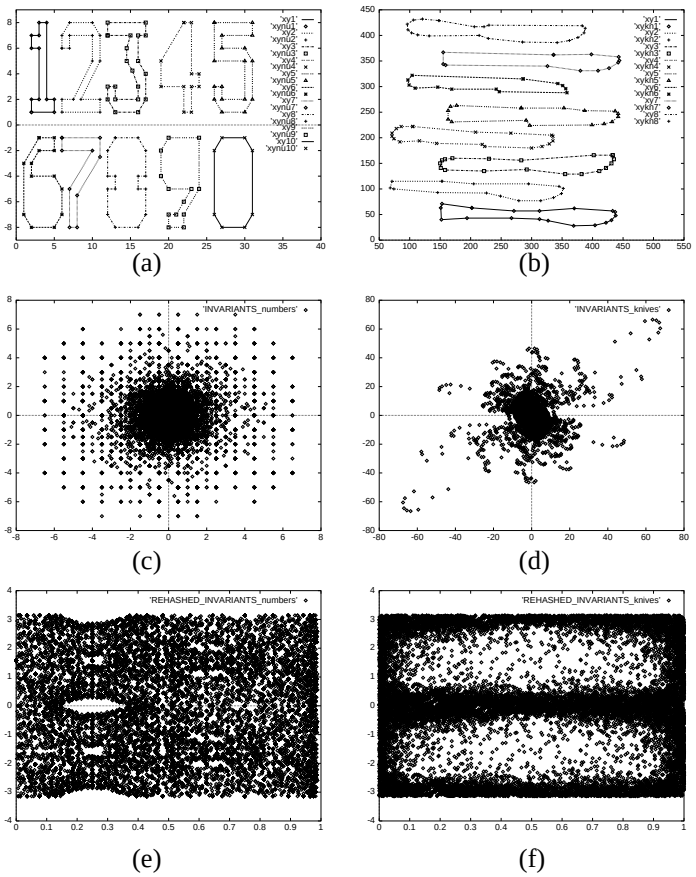

'xy1'
'xyu1'
'xy2'
'xyu2'
'xy3'
'xyu3'
'xy4'
'xyu4'
'xy5'
'xyu5'
'xy6'
'xyu6'
'xy7'
'xyu7'
'xy8'
'xyu8'
'xy9'
'xyu9'
'xy10'
'xyu10'
(a)
'xy1'
'xykh1'
'xy2'
'xykh2'
'xy3'
'xykh3'
'xy4'
'xykh4'
'xy5'
'xykh5'
'xy6'
'xykh6'
'xy7'
'xykh7'
'xy8'
'xykh8'
(b)
'INVARIANTS_numbers'
(c)
'INVARIANTS_knives'
(d)
'REHASHED_INVARIANTS_numbers'
(e)
'REHASHED_INVARIANTS_knives'
(f)

Figure 9

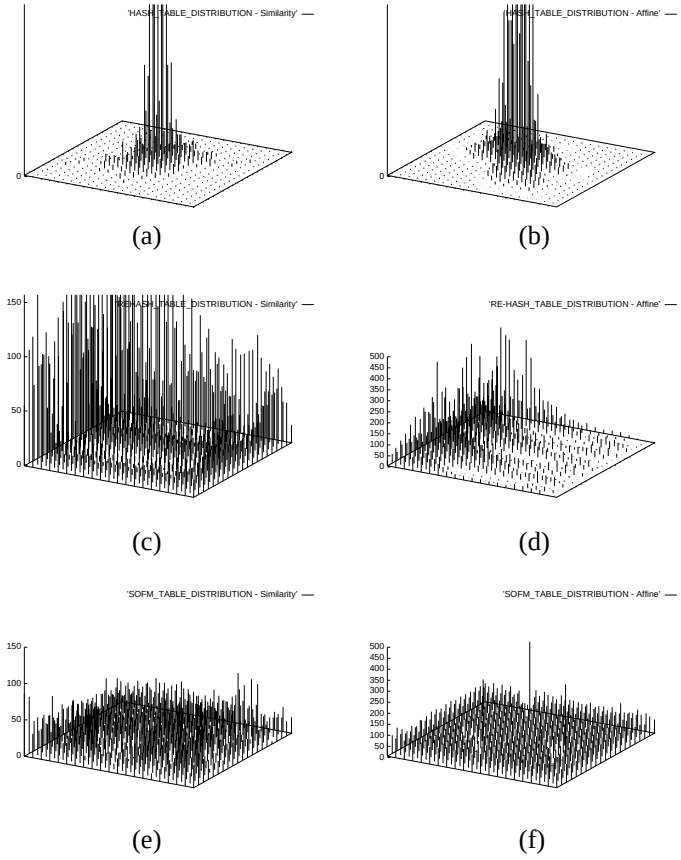
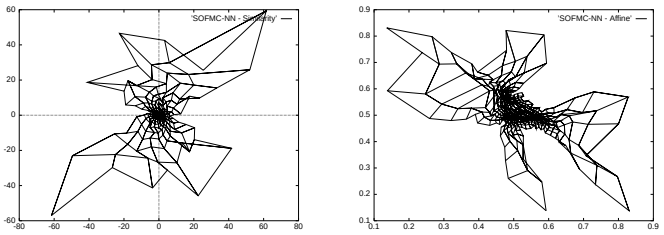
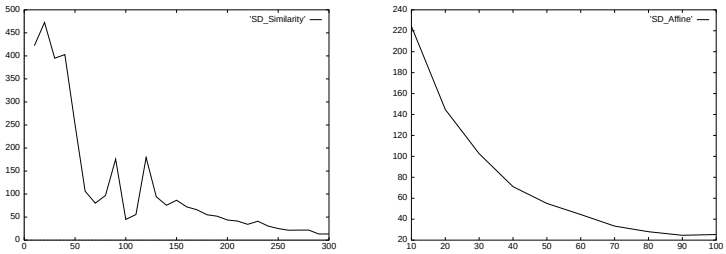

'HASH_TABLE_DISTRIBUTION - Similarity'
0
(a)
'HASH_TABLE_DISTRIBUTION - Affine'
0
(b)
'RE-HASH_TABLE_DISTRIBUTION - Similarity'
150
100
50
0
(c)
'RE-HASH_TABLE_DISTRIBUTION - Affine'
500
450
400
350
300
250
200
150
100
50
0
(d)
'SOFM_TABLE_DISTRIBUTION - Similarity'
150
100
50
0
(e)
'SOFM_TABLE_DISTRIBUTION - Affine'
500
450
400
350
300
250
200
150
100
50
0
(f)

Figure 10



(a) (b)

Figure 11



(a) (b)

Figure 12

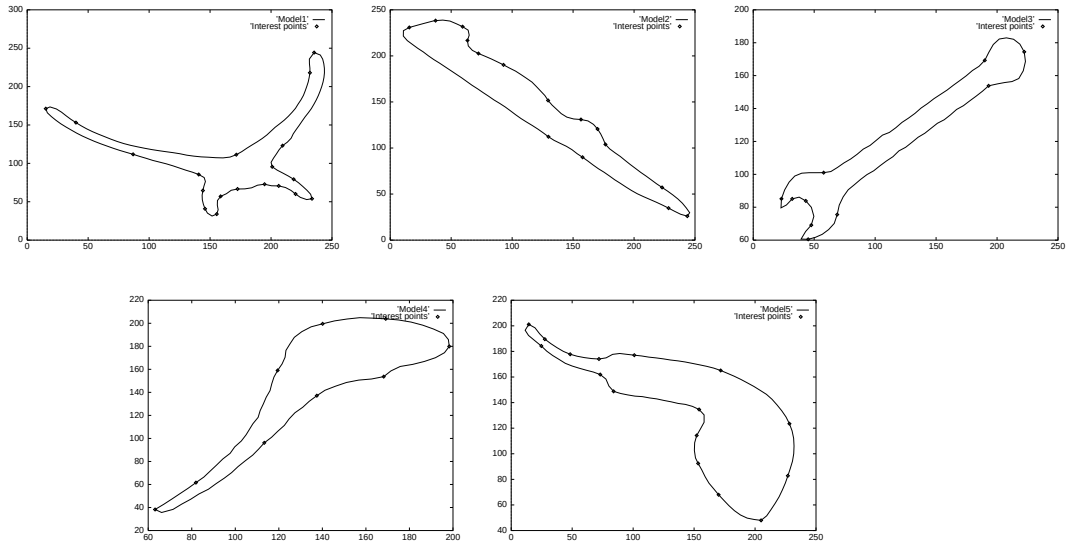
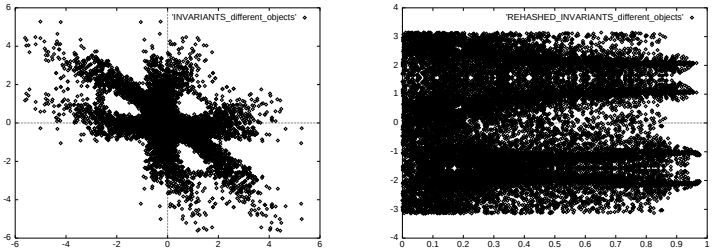

'Model1'
'Interest points'
'Model2'
'Interest points'
'Model3'
'Interest points'
'Model4'
'Interest points'
'Model5'
'Interest points'

Figure 13


'INVARIANTS_different_objects'
'REHASHED_INVARIANTS_different_objects'

(a) (b)

Figure 14

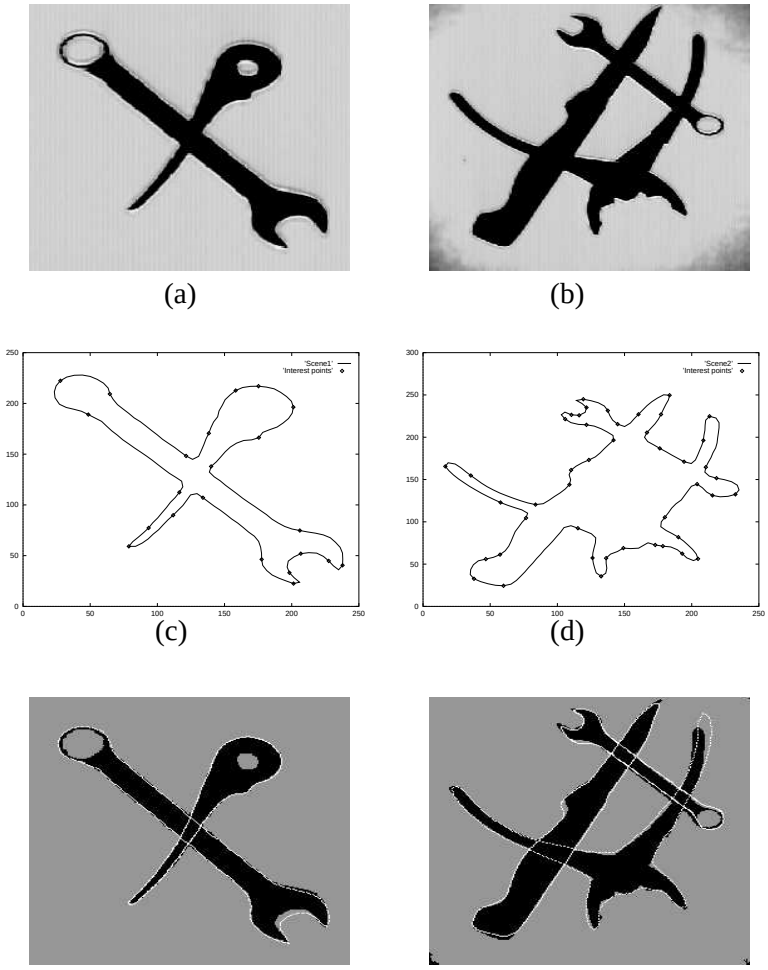


Figure 15