Learning Affine Transformations

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

correspondence should be addressed to

Dr. George Bebis
Department of Computer Science
University of Nevada
Reno, NV 89557

E-mail: bebis@cs.unr.edu
Tel: (702) 784-6463
Fax: (702) 784-1877

Abstract

Under the assumption of weak perspective, two views of the same planar object are related through an affine transformation. In this paper, we consider the problem of training a simple neural network to learn to predict the parameters of the affine transformation. Although the proposed scheme has similarities with other neural network schemes, its practical advantages are more profound. First of all, the views used to train the neural network are not obtained by taking pictures of the object from different viewpoints. Instead, the training views are obtained by sampling the space of affine transformed views of the object. This space is constructed using a single view of the object. Fundamental to this procedure is a methodology, based on Singular Value Decomposition (SVD) and Interval Arithmetic (IA), for estimating the ranges of values that the parameters of affine transformation can assume. Second, the accuracy of the proposed scheme is very close to that of a traditional least squares approach with slightly better space and time requirements. A front-end stage to the neural network, based on Principal Components Analysis (PCA), shows to increase its noise tolerance dramatically and also to guides us in deciding how many training views are necessary in order for the network to learn a good, noise tolerant, mapping. The proposed approach has been tested using both artificial and real data.

Keywords: object recognition, object recognition, artificial neural networks

1. Introduction

Affine transformations have been widely used in computer vision and particularly, in the area of model-based object recognition [1]-[5]. Specifically, they have been used to represent the mapping from a 2-D object to a 2-D image or to approximate the 2-D image of a planar object in 3-D space and it has been shown that a 2-D affine transformation is equivalent to a 3-D rigid motion of the object followed by orthographic projection and scaling (weak perspective). Here, we consider the case of real planar objects, assuming that the viewpoint is arbitrary. Given a known and an unknown view of the same planar object, it is well known that under the assumption of weak perspective projection ([1][2]), the two views are related through an affine transformation. Given the point correspondences between the two views, the affine transformation which relates the two views can be computed by solving a system of linear equations using a least-squares approach (see section 3).

In this paper, we propose an alternative approach for computing the affine transformation based on neural networks. The idea is to train a neural network to predict the parameters of the affine transformation using the image coordinates of the points in the unknown view. A shorter version of this work can be found in [7]. There two main reasons which motivated us in using neural networks to solve this problem. First of all, it is interesting to think of this problem as a learning problem. There have also been proposed several other approaches ([11][12]) which treat similar problems as learning problems. Some of the issues that must be addressed within the context of this formulation are: (i) how to obtain the training views, (ii) how many training views are necessary, (iii) how long it takes for the network to learn the desired mapping, and (iv) how accurate are the predictions. Second, we are interested in comparing the neural network approach with traditional least-squares used in the computation of the affine transformation. Given that neural networks are inherently parallelizable, the neural network approach might be a good alternative if it turns out that it is as accurate as traditional least squares approaches. In fact, our experimental results demonstrate that the accuracy of the neural network scheme is as good as that of traditional least-squares with the proposed approach having slightly less space and time requirements.

There are three main steps in the proposed approach. First, the ranges of values that the parameters of affine transformation can assume are estimated. We have developed a methodology based on Singular Value Decomposition (SVD) [8] and Interval Arithmetic (IA) [9] for this. Second, the space of parameters is

sampled. For each set of sampled parameters, an affine transformation is defined which is applied on the known view to generate a new view. We will be referring to these views as *transformed views*. The transformed views are then used to train a Single Layer Neural Network (SL-NN) [10]. Given the image coordinates of the points of the object in the transformed view, the SL-NN learns to predict the parameters of the affine transformation that align the known and unknown views. After training, the network is expected to *generalize*, that is, to be able to predict the correct parameters for transformed views that were never exposed to it during training.

The proposed approach has certain similarities with two other approaches [11][12]. In [11], the problem of approximating a function that maps any perspective 2-D view of a 3-D object to a standard 2-D view of the same object was considered. This function is approximated by training a Generalized Radial Basis Functions Neural Network (GRBF-NN) to learn the mapping between a number of perspective views (training views) and a standard view of the model. The training views are obtained by sampling the viewing sphere, assuming that the 3-D structure of the object is available. In [12], a linear operator is built which distinguishes between views of a specific object and views of other objects (orthographic projection is assumed). This is done by mapping every view of the object to a vector which uniquely identifies the object. Obviously, our approach is conceptually similar to the above two approaches, however, there are some important differences. First of all, our approach is different in that it does not map different views of the object to a standard view or vector but it computes the parameters of the transformation that align known and unknown views of the same object. Second, in our approach, the training views are not obtained by taking different pictures of the object from different viewpoints. Instead, they are affine transformed views of the known view. On the other hand, the other approaches can compute the training views *easily* only if the structure of the 3-D object is available. Since this is not always available, the training views can be obtained by taking different pictures of the object from various viewpoints. However, this requires more effort and time (edges must be extracted, interest point must be identified, and point correspondences across the images must be established). Finally, our approach does not consider both the x- and y-coordinates of the object points during training. Instead, we simplify the scheme by decoupling the coordinates and by training the network using only one of the two (the x-coordinates here). The only overhead from this simplification is that the parameters of the affine transformation must be

computed in two steps.

There are two comments that should be made at this point. First of all, the reason that a SL-NN is used is because the mapping to be learned is linear. This should not be considered, however, as a trivial task since both the input (image) and output (parameter) spaces are continuous. In other words, special emphasis should be given on the training of the neural network to ensure that the accuracy in the predictions is acceptable. Second, it should be clear that the proposed approach assumes that the point correspondences between the unknown and known views of the object are given. That was also the case with [11] and [12]. Of course, establishing the point correspondences between the two views is the most difficult part in solving the recognition problem. Unless the problem to be solved is very simple, using the neural network approach without any *a-priori* knowledge about possible point correspondences is not efficient in general (see [20][21] for some example). On the other hand, combining the neural network scheme with an approach which returns possible point correspondences will be ideal. For example, we have incorporated the proposed neural network scheme in an indexing-based object recognition system [15]. In this system, groups of points are chosen from the unknown view and are used to retrieve hypotheses from a hash table. Each hypothesis contains information about a group of object points as well as information about the order of the points in the group. This information can be used to place the points from the unknown view into a correct order before they are fed to the network.

There are various issues to be considered in evaluating the proposed approach such as, how good is the mapping computed by the SL-NN, what is the discrimination power of the SL-NNs, and how accurate are the predictions of the SL-NN assuming noisy and occluded data. These issues have been considered in section 5. The quality of the approximated mapping depends rather on the number of training views used to train the neural network. The term "discrimination power" means the capability of a network to predict wrong transformation parameters, assuming that it is exposed to views which belong to different objects than the one whose views were used to train the network (*model specific networks*). Our experimental results show that the discrimination power of the networks is very good. Testing the noise tolerance of the the networks, we found that it was rather poor. However, we were able to account for it by attaching a front-end stage to the inputs of the SL-NN. This stage is based on Principal Components Analysis (PCA) [16] and its benefits are very important.

Our experimental results show a dramatic increase in the noise tolerance of the SL-NN. We have also noticed some improvements in the case of occluded data, but the performance degrades rather rapidly even with 2-3 points missing. In addition, it seems that PCA can guide us in deciding how many training views are necessary in order for the SL-NN to learn a "good", noise tolerant, mapping.

The organization of the paper is as follows: Section 2 presents a brief review of the affine transformation. Section 3 presents the procedure for estimating the ranges of values that the parameters of the affine transformation can assume. In Section 3, we describe the procedure for the generation of the training views and the training the SL-NNs. Our experimental results are given in Section 4 while our conclusions are given in Section 5.

2. Affine transformations

Let us assume that each object is characterized by a list of "interest" points $(p_1', p_2', \cdots, p_m')$, which may correspond, for example, to curvature extrema or curvature zero-crossings [6]. Let us now consider two images of the same planar object, each one taken from a different viewpoint, and two points $p = (x, y)$, $p' = (x', y')$, one from each image, which are in correspondence; then the coordinates of p can be expressed in terms of the coordinates of p', through an affine transformation, as follows:

$$p = Ap' + b \tag{1}$$

where A is a non-singular 2 x 2 matrix and b is a two-dimensional vector. A planar affine transformation can be described by 6 parameters which account for translation, rotation, scale, and shear. Writing (1) in terms of the image coordinates of the points we have:

$$x = a_{11}x' + a_{12}y' + b_1 \tag{2}$$

$$y = a_{21}x' + a_{22}y' + b_2 \tag{3}$$

The above equations imply that given two different views of an object, one known and one unknown, the coordinates of the points in the unknown view can be expressed as a linear combination of the coordinates of

the corresponding points in the known view. Thus, given a known view of an object, we can generate new, affine transformed views of the same object by choosing various values for the parameters of the affine transformation. For example, Figures 1(b)-(d) show affine transformed views of the planar object shown in Figure 1(a). These views were generated by transforming the known view using the affine transformations shown in Table 1. Thus, for any affine transformed view of a planar object, there is a point in the 6-dimensional space of 2-D affine transformations which corresponds to the transformation that can bring the known and unknown views into alignment (in a least-squares sense).

<Figure 1. -- about here>

<Table 1. -- about here>

3. Estimating the ranges of the parameters

Given a known view I' and an unknown affine transformed view I of the same planar object, as well as the point correspondences between the two views, there is an affine transformation that can bring I' into alignment with I. In terms of equations, this can be written as follows:

$$I' \begin{bmatrix} A \\ b \end{bmatrix} = I \tag{4}$$

or

$$\begin{bmatrix} x_1' & y_1' & 1 \\ x_2' & y_2' & 1 \\ \cdots & \cdots & \cdots \\ x_m' & y_m' & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ b_1 & b_2 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \cdots & \cdots \\ x_m & y_m \end{bmatrix} \tag{5}$$

where $(x_1, y_1), (x_2, y_2), \cdots (x_m, y_m)$ are the coordinates of the points corresponding to I, while $(x_1', y_1'), (x_2', y_2'), \cdots (x_m', y_m')$ are the coordinates of the points corresponding to I'. We assume that both views consist of the same number of points. To achieve this, we consider only the points that are common in both views. Equation (5) can be split into two different systems of equations, one for the x-coordinates and one for the y-coordinates of I, as follows:

$$\begin{bmatrix} x_1' & y_1' & 1 \\ x_2' & y_2' & 1 \\ \cdots & \cdots & \cdots \\ x_m' & y_m' & 1 \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ b_1 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_m \end{bmatrix} \tag{6}$$

$$\begin{bmatrix} x_1' & y_1' & 1 \\ x_2' & y_2' & 1 \\ \cdots & \cdots & \cdots \\ x_m' & y_m' & 1 \end{bmatrix} \begin{bmatrix} a_{21} \\ a_{22} \\ b_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_m \end{bmatrix} \tag{7}$$

Using matrix notation, we can rewrite (6) and (7) as $Pc_1 = p_x$ and $Pc_2 = p_y$ correspondingly, where P is the matrix formed by the x- and y-coordinates of the points of the known view I', c_1 and c_2 represent the parameters of the transformation, and p_x, p_y, are the x- and y-coordinates of the unknown view I. Both (6) and (7) are over-determined (the number of points is usually larger than the number of parameters, that is, $m \gg 3$), and can be solved using a least-squares approach such as SVD [8]. Using SVD, we can factor the matrix P as follows:

$$P = UWV^T \tag{8}$$

where both U and V are orthogonal matrices (m x 3 and 3 x 3 size correspondingly), while W is a diagonal matrix (3 x 3 size) whose elements w_{ii} are always non-negative and are called the singular values of P. The solutions of (6) and (7) are then given by $c_1 = P^+ p_x$ and $c_2 = P^+ p_y$, where P^+ is the pseudo-inverse of P which is equal to $P^+ = VW^+U^T$, where W^+ is also a diagonal matrix with elements $1/w_{ii}$, if w_{ii} greater than zero (or a very small threshold in practice), and zero otherwise. Taking this into consideration, the solutions of (6) and (7) are given by ([15]):

$$c_1 = \sum_{i=1}^{3} \left(\frac{u_i^T p_x}{w_{ii}}\right) v_i \tag{9}$$

$$c_2 = \sum_{i=1}^{3} \left(\frac{u_i^T p_y}{w_{ii}}\right) v_i \tag{10}$$

where u_i denotes the i-th column of matrix U and v_i denotes the i-th column of matrix V. Of course, the sum should be restricted for those values of i for which $w_{ii} \neq 0$.

To determine the range of values for the parameters of affine transformation, we first assume that the image of the unknown view has been scaled so that the x- and y-coordinates of the object points belong within a specific interval. This can be done, for example, by mapping the image of the unknown view to the unit square. In this way, its x- and y-coordinates are mapped to the interval [0, 1]. To determine the range of values for the parameters of affine transformation, we need to consider all the possible solutions of (6) and (7), assuming that the components of the vectors in the right hand side of the equations are always restricted to belong to the interval [0,1]. Trying to calculate the range of values using mathematical inequalities did not yield "good" results in the sense that the novel views corresponding to the ranges computed were not spanning the whole unit square but only a much smaller sub-square within it. Therefore, we consider Interval Arithmetic [9]. In IA, each variable is actually represented as an interval of possible values. Given two interval variables $t = [t_1, t_2]$ and $r = [r_1, r_2]$, then the sum and the product of these two interval variables is defined as follows:

$$t + r = [t_1 + r_1, t_2 + r_2] \tag{11}$$

$$t * r = [min(t_1 r_1, t_1 r_2, t_2 r_1, t_2 r_2), max(t_1 r_1, t_1 r_2, t_2 r_1, t_2 r_2)] \tag{12}$$

Obviously, variables which assume only fixed values can still be represented as intervals, trivially though, by considering the same fixed value for both left and right limits. Applying interval arithmetic operators to (9) and (10) instead of the standard arithmetic operators, we can compute interval solutions for c_1 and c_2 by setting p_x and p_y equal to [0,1]. In interval notation, we want to solve the systems $Pc_1^I = p_x^I$ and $Pc_2^I = p_y^I$, where the superscript I denotes an interval vector. The solutions c_1^I and c_2^I should be understood to mean $c_1^I = [c_1 \colon Pc_1 = p_x,\ p_x \in p_x^I]$ and $c_2^I = [c_2 \colon Pc_2 = p_y,\ p_y \in p_y^I]$. It should be noted that since both interval systems involve the same matrix P and p_x, p_y assume values in the same interval, the interval solutions c_1^I and c_2^I will be the same. For this reason, we consider only the first of the interval systems in our analysis.

A lot of research has been done in the area of interval linear systems [17]. In more complicated cases,

the matrix of the system of equations is also an interval matrix, that is, a matrix whose components are interval variables. Our case here is simpler, since the elements of P_{xy} are the x- and y-coordinates of the known object view which are fixed. However, if we merely try to evaluate (9) using the interval arithmetic operators described above, most likely we will obtain a non-sharp interval solution. The concept of non-sharp interval solutions is very common in IA. When we solve interval systems of equations, not all of the solutions obtained satisfy the problem at hand [17][18]. We will be referring to these solutions as invalid solutions. An interval solution is considered to be sharp if it includes as few invalid solutions as possible. The reason that sharp interval solutions are very desirable in our approach is because the generation of the training views can be performed faster (see next section). The sharpness of the solutions obtained using IA depends on various factors. One well known factor that affects sharpness is when a given interval quantity enters into a computation more than once [18]. This is actually the case with (9). To make it clear, let us consider the solution for the $i-th$ component of c_1, $1 \le i \le 3$:

$$c_{i1} = \frac{v_{i1}}{w_{11}}(u_{11}x_1 + u_{21}x_2 + \cdots + u_{m1}x_m) + \frac{v_{i2}}{w_{22}}(u_{12}x_1 + u_{22}x_2 + \cdots + u_{m2}x_m) + \frac{v_{i3}}{w_{33}}(u_{13}x_1 + u_{23}x_2 + \cdots + u_{m3}x_m) \quad (13)$$

Clearly, each x_j $(1 \le j \le m)$ enters in the computations of c_{i1} more than once. To avoid this, we factor out the x_j's. Then, (13) takes the form:

$$c_{i1} = \sum_{j=1}^{m} x_j \left(\sum_{k=1}^{3} \frac{v_{ik}u_{jk}}{w_{kk}}\right) \quad (14)$$

<Figure 2. -- about here>

The interval solution of c_{i1} can now be obtained by applying interval arithmetic operators in (14) instead of (13). Similarly, we can obtain interval solutions for the remaining elements of c_1^I as well as for c_2^I. It should be mentioned that given the solutions c_1^I and c_2^I, then $p_x^I \subseteq Pc_1^I$ and $p_y^I \subseteq Pc_2^I$. In other words, not every solution in c_1^I and c_2^I corresponds to p_x and p_y that belong in p_x^I and p_y^I respectively. This issue is further

discussed in the next section.

4. Learning the mapping

In order to train the SL-NN, we first need to generate the training views. This is performed by sampling the space of affine transformed views of the object. This space can be constructed by transforming a known view of the object, assuming all the possible sets of values for the parameters of affine transformation. Since it is impossible to consider all the possible sets, we just sample the range of values of each parameter and we consider only a finite number of sets. However, it is important to keep in mind that not all of the invalid solutions contained in the interval solutions of (9) might have been eliminated completely. As a result, when we generate affine transformed views by choosing the parameters of affine transformation from the interval solutions obtained, then not all of the generated views will lie in the unit square completely (invalid views). These views correspond to invalid solutions and must be disregarded. Figure 2(a) illustrates the procedure. It might be clear now why it is desirable to compute sharp interval solutions. Sharp interval solutions imply narrower ranges for the parameters of affine transformation and consequently, the sampling procedure of Figure 2(a) can be implemented faster.

<Figure 3. -- about here>

It is important to observe at this point that both equations for computing x_i and y_i ((2) and (3) which appear in Figure 2(a)) involve the same basis vector (x', y'). Also, given that the ranges of (a_{11}, a_{12}, b_1) will be the same with the ranges of (a_{21}, a_{22}, b_2), as we discussed in section 3, the information to be generated for the x_i coordinates will be exactly the same as the information to be generated for the y_i coordinates. Hence, we decouple the x- and y-coordinates of the views and we generate information only for one of the two (the x-coordinates here). This is illustrated in Figure 2(b). This observation offers great simplifications since the sampling procedure shown in Figure 2(a) can now take a much more simplified form as shown in Figure 2(b). Consequently, the time and space requirements of the procedure for generating and storing the training views are significantly reduced. Furthermore, the size of the SL-NN is reduced in half. Assuming $\bar{m}$ interest points per view on the average, the sampling scheme of Figure 2(a) requires a network with $2\bar{m}$ input

nodes and 6 output nodes (Figure 3(a)) while the sampling scheme of Figure 2(b) requires only $\bar{m}$ input nodes and 3 output nodes (Figure 3(b)). It should be noted that although we consider only one of the two image point coordinates of the training views, we are still referring to them as training views and this should not cause any confusion.

<Figure 4. -- about here>

The decoupling of the point coordinates of the views and the consideration of only one of the two, imposes an additional cost during the recovery of the transformation parameters: they must now be predicted in two steps: First, we need to feed to the network the x-coordinates of the unknown view in order to predict (a_{11}, a_{12}, b_1) and then we need to feed to the network the y-coordinates to predict (a_{21}, a_{22}, b_2). However, given the fast response time of the SL-NN after training has been completed, this additional cost is not very important. Figure 4 presents an overview of the procedure for training a SL-NN to approximate the mapping between the image coordinates of an object's points and the space of parameters of the affine transformation. The meaning of the box labeled "PCA" will be discussed later.

5. Experiments

In this section, we report a number of experiments in order to demonstrate the strengths and weakness of the proposed approach. We have considered various issues such as accuracy in the predictions, discrimination power, and tolerance to noisy and occluded data.

<Figure 5. -- about here >

5.1. Evaluation of the SL-NNs' performance

First, we evaluated how "good" the mapping computed by the SL-NN is. The following procedure was applied: first, we generated random affine transformed views of the object by choosing random values for the parameters of affine transformation. Then, we normalized the generated affine transformed views so that their x- and y-coordinates belong to the unit square. This was performed by choosing a random sub-square within the unit square and by mapping the square enclosing the view of the object (defined by its minimum and

maximum x- and y-coordinates) to the randomly chosen sub-square. After normalization, we applied the x-coordinates of the normalized unknown view first, and then its y-coordinates, to the SL-NN in order to predict the affine transformation that can align the known view with the normalized unknown view.

To judge how good the predictions yielded by the SL-NN were, we performed two tests: First, we compared the predicted values for the parameters of the affine transformation with the actual values which were computed using SVD. Second, we computed the mean square error between the normalized unknown view of the object and the back-projected known view, which was obtained by simply applying the predicted affine transformation on the known view. This is the most commonly used test in hypothesis generation - verification methods [1][2]. Figure 5 summarizes the evaluation procedure. Figure 6 shows the four different objects used in our experiments. For each object, we have identified a number of boundary "interest" points, which correspond to curvature extrema and zero-crossings [6]. These points are also shown in Figure 6. The training of the SL-NN is based only on the coordinates of these "interest" points, however, the computation of the mean square error between the back-projected view and the unknown view utilizes all the boundary points for better accuracy. First, we estimated for each object the ranges of values that the parameters of affine transformation can assume. Only the interest points of each object were used for this estimation. Table 2 shows the ranges computed.

<Figure 6. -- about here>

<Table 2. -- about here>

For each object, we generated a number of training views by sampling the space of affine transformed views of the object and we trained a SL-NN to learn the desired mapping. One layer architectures were used because the mapping to be approximated is linear. The number of nodes in the input layer was determined by the number of interest points associated with each object while the number of nodes in the output layer was set to three (equal to the three parameters a_{11}, a_{12}, and b_1). Linear activation functions were used for the nodes in the output layer. Training was performed using the back-propagation algorithm [10]. Back-propagation is an iterative algorithm which in each step adjusts the connection weights in the network, minimizing an error function. This is achieved using a gradient search which corresponds to a steepest descent on

an error surface representing the weight space. The weight adjustment is determined by the current error and a parameter called *learning rate* which determines what amount of the error sensitivity to weight change will be used for the weight adjustment. In this study, a variation of the back-propagation algorithm (back-propagation with momentum) was used [10]. This is a simple variation for speeding up the back-propagation algorithm. The idea is to give each weight change some momentum so that it accelerates in the average down-hill direction. This may prevent oscillations in the system and help the system escape local error function minima. It is also a way of increasing the effective learning rate in almost-flat regions of the error surface. In all of our experiments, we used the same learning rate (0.2) and the same momentum term (0.4). We assumed that the network had converged when the sum of squared errors between the desired and actual outputs was less than 0.0001. Larger values (˜ 0.01) can still lead to a well trained network, however, we found that the network becomes more sensitive to noise if we choose a more relaxed stopping criterion.

<Table 3. -- about here>

Table 3 shows some affine transformations predicted by a network trained with only 4 training views for the case of Model1. These views were generated by sampling each parameter's range at 6 points. Views with image coordinates outside the interval [0,1] were not considered as training views, according to our discussion in section 4. This is why although we sampled each parameter at six points, we finally ended up with only four training views. The actual parameters are also shown for comparison purposes. In addition, Table 3 shows the parameters predicted, for the same set of test affine transformed views, by a network trained with 73 views which were generated by sampling each parameter's range at 15 points. It can be observed that the predictions made by the network trained with the 73 views are not significantly better than the predictions made by the network trained with the 4 views.

<Table 4. -- about here>

Table 4 presents results for all of the planar objects, using various numbers of training views. For each case, we report the number of samples per parameter's range and the generated number of training views. Since it is not very easy to evaluate the quality of the predictions by simply examining the predicted

parameter values, we also report the average mean square back-projection error and standard deviation. These were computed using 100 randomly transformed views for each object. Also, to get an idea of the training time, we report the number of training epochs required for convergence. These results indicate that the SL-NN is capable of approximating the desired mapping very accurately, it does not require many training views, and training time is fast. Increasing the number of training views did not yield a significant improvement in the case of noise-free data.

We also examined the computational requirements of the neural network approach. In our comparison, we assume that the training of the network is done off-line. If $\bar{m}$ is the average number of interest points per model, the neural network approach requires $3\bar{m}$ multiplications and $3\bar{m}$ additions to predict a_{11}, a_{12} and b_1. The same number of operations are required for predicting the other three parameters, so it requires $6\bar{m}$ multiplications and $6\bar{m}$ additions totally. For comparison, we also examined the computational requirements of a traditional least-squares approach. Specifically, we chose the SVD approach. Assuming that the factorization of P_{xy} is also done off-line, SVD requires $12\bar{m}$ multiplications, $6\bar{m}$ divisions, and $6(\bar{m}+6)$ additions. Given that these computations are repeated hundred of times during verification in object recognition, the neural network approach turns out to have less computational requirements. Also, the neural network approach has lower memory requirements than the traditional approach. Specifically, the neural network approach requires to store only $6\bar{m}$ values per network (i.e., weights) while the traditional approach requires to store $6\bar{m} + 6 + 6^2$ values (for the elements of U, W, and V matrices). To avoid confusion, we need to emphasize again that the above comparison assumes that training and decomposition have been performed off-line. When this assumption is not true, then the SVD approach is faster than the neural network approach.

5.2. Discrimination power

Next, we investigated the discrimination power of each of the networks. For each object, we used the SL-NN trained with the numbers of training views shown highlighted in Table 4. These networks are noise tolerant and require a minimum number of training views to learn the mapping. Since each neural network has a different number of input nodes, depending on the number of interest points associated with the objects, it is practically impossible to present views of different objects, with different number of interest points, to the

same network. To overcome this problem, we have attached a front-end stage to the SL-NN which actually reduces the dimensionality of the input vector, formed by the coordinates of the interest points of the views. In this way, we could use the same network architecture for each object. The front-end stage is based on Principal Components Analysis (PCA) [16]. PCA is a multivariate technique which transforms a number of correlated variables, to a smaller set of uncorrelated variables. PCA might have important benefits for the performance of the neural network since less inputs, which are also uncorrelated, imply faster training and probably better generalization. PCA works as follows: first, we compute the covariance matrix associated with our correlated variables and then we find the eigenvalues of this matrix. Then, we sort them and we form a new matrix whose columns consist of the eigenvectors to the largest eigenvalues. Deciding how many eigenvalues are significant depends on the problem at hand. The matrix formed by the eigenvectors corresponds to the transformation which is applied on the correlated variables to yield the new uncorrelated variables.

In our problem, the correlated variables are the training views associated with each SL-NN. For each training set, we applied the PCA and we kept the most significant principal components, three principal components were kept since only three eigenvalues were non-zero. The new training examples are now linear combinations of the old training views with dimensionality three. A separate network per object was used, having 3 nodes in the input layer and 3 nodes in the output layer. After training, we tested each network's discrimination ability. The results (average back-projection error and standard deviation over 100 randomly chosen affine transformed views for each model) are presented in Table 5. Clearly, each network predicts the correct affine transformation only for the affine transformed views of the object whose views were used to train the network. The discrimination power of the networks can be very useful during recognition. For example, suppose that we are given an unknown view. In order to recognize the object which has produced this view, it suffices to present the view to all of the networks. Each network will predict a set of transformation parameters, however, only one network (corresponding to the object which has produced the unknown view) will predict the correct parameters.

<Table 5. -- about here>

5.3. Noise tolerance

In this subsection, we investigate how tolerant the networks' predictions are, assuming uncertainty in the locations of the object points. In particular, we assume that the location of each object point can be anywhere within a disc centered at the real location of the point and having a radius equal to ε (bounded uncertainty) [19]. Various ε values were chosen in order to evaluate the networks' ability to predict the correct transformation parameters. To test the networks, we used a set of 100 random affine transformed views and we computed the average mean square back-projection error. The results obtained, assuming that the front-end stage is inactive, show that the performance of the networks is rather poor. Figure 7 (solid lined) shows a plot of the average mean square back-projection error versus ε. Also, we show the minimum and maximum errors observed. Trying to improve performance by using more training views did not help significantly. For instance, assuming $\varepsilon = 0.2$ and 4 training views for Model1 (first row in Table 4) resulted in a mean square back-projection error equal to 1.622 with a standard deviation equal to 1.692. Assuming the same value for ε and 14 views, resulted in a mean square back-projection error equal to 1.62 with a standard deviation equal to 1.69. Using more views did not yield much better results.

<Figure 7. -- about here>

Then, we tested the performance of the networks, assuming that the front-end stage is active. What we observed is quite interesting. For a small number of training views, the performance was not significantly better than the performance obtained using the SL-NNs trained with the original views (i.e., having the front-end stage inactive). However, a dramatic increase in the noise tolerance was observed by training the SL-NNs using more views. For instance, assuming Model1, $\varepsilon = 0.2$ and 4 training views, resulted in a mean square back-projection error equal to 1.659 with a standard deviation equal to 1.39. These results are slightly better than those obtained using the original training views. However, assuming the same ε value and 14 views, resulted in a mean square back-projection error equal to 0.338 with a standard deviation equal to 0.224, a dramatic decrease. Figure 7 (dashed line) shows a plot of the average mean square back-projection error vs ε, as well as the minimum-maximum errors observed in this case. Some specific examples are shown in Figure 8, where the figures in the left column show the matches achieved without using the PCA front-end stage, while

the figures in the right column show the matches achieved using the PCA front-end stage. The solid line represents the unknown view and the dashed line represents the back-projected view which was computed using the predicted parameters. The actual and predicted parameters are shown in Table 6.

<Figure 8. -- about here>

<Table 6. -- about here>

In particular, we observed that in the cases where the number of training views was not enough for the network to be noise tolerant, the number of non-zero eigenvalues associated with the covariance matrix of the training views was consistently less than three. Assuming more training views did not improve noise tolerance as long as the number of non-zero eigenvalues was less than three. However, utilizing enough training views so that the number of non-zero eigenvalues was three, resulted in a dramatic error decrease. Including more training views after this point did not improve noise tolerance significantly, and the number of non-zero eigenvalues remained three. The same observations were made for all of the four objects used in our experiments. The reason we finally end up with three non-zero eigenvalues is related to the fact that only three points are necessary to compute the parameters of the affine transformation. On the other hand, the training views obtained by sampling the space of transformed views might not span the space satisfactorily because of degenerate views. However, PCA can guide us in choosing a sufficient number of training views so that the networks can compute good, noise tolerant, mappings.

5.4. Occlusion tolerance

We have also performed a number of experiments assuming that some points are occluded. The performance of the SL-NNs trained with the original views was extremely bad, even with one point missing. Incorporating the PCA front-end stage improved the performance in cases where 2-3 points were missing. However, the performance was still poor when more points were removed. This suggests that in order for someone to apply the proposed method in cases where data occlusion is present, training of different networks for each object, using subsets of points rather than on all the object points, is more appropriate. The simplest way to select subsets of points is randomly. This, however, is not very efficient since the number of subsets increases

exponentially with the number of points. A more efficient approach would be to apply a grouping approach [22][23] to detect groups of points which belong to a particular object.

5.5. Performance using real scenes

In this section, we demonstrate the performance of the method using real scenes. Figure 9(a),(b) shows two of the scenes used in our experiments. The first scene contains Model1, Model2, and Model3 while the second scene contains Model1 and Model4 as well as another object that we have not considered in our experiments. In Scene1, we have intentionally left out the inner contour to make recognition more difficult. Point correspondences were established by hand. In cases that a model point did not have an exact corresponding scene point, we chose the closest possible scene point. Also, in cases that a model point did not have a corresponding scene point because of occlusion (for example, 1 point is occluded in Model1 in Scene1 and 2 points are occluded in Model2 in Scene1), we just picked the point (0.5, 0.5) (the center of the unit square) to be the corresponding scene point. The models were back-projected onto the scenes using the parameters predicted by the networks. The results are shown in Figure 9(e),(f). As it can be seen, the models present in the scene have been recognized and aligned fairly well with the scene. It should be noted that in addition to the noise we have introduced by substituting missing "interest" points by neighboring points or even artificial points, there is also noise in the location of of the rest scene points due to lack of robustness in the edge detection or/and "interest" point extraction. The best alignment was achieved in the case of Model1 and Model3 where most of their interest points were visible. The alignment of Model2 has some problems at the non-sharp end of the object because there were missing "interest" points in this area as well as noise in the location of the rest points. Finally, Model4 has been aligned with the scene quite satisfactorily. In the area of the boundary where the alignment is not very good, there was an "interest" point which was not detected and thus was replaced by the point (0.5, 0.5) in the prediction of the affine transformation.

<Figure 9. -- about here>

6. Conclusions

In this paper, we considered the problem of learning to predict the parameters of the transformation that can align a known view of an object with unknown views of the same object. Initially, we compute the possible range of values that the parameters of the alignment (affine) transformation can assume. This is performed using Singular Value Decomposition (SVD) and Interval Analysis (IA). Then, we generate a number of novel views of the object by sampling the space of its affine transformed views. Finally, we train a Single Layer Neural Network (SL-NN) to learn the mapping between the affine transformed views and the parameters of the alignment transformation. A number of issues related to the performance of the neural networks were considered such as accuracy in the predictions, discrimination power, noise tolerance, and occlusion tolerance.

Although our emphasis in this paper is to study the case of planar objects and affine transformations, it is important to mention that the same methodology can be extended and applied to the problem of learning to recognize 3-D objects from 2-D views, assuming orthographic or perspective projection. The linear model combinations scheme proposed by Basri and Ullman [12] and the algebraic functions of views proposed by Shashua [13] serve as a basis for this extension. In this case, novel orthographic or perspective views can be obtained by combining the image coordinates of a small number of reference views instead of a single reference view. The training views can be obtained by sampling the space of orthographically or perspectively transformed views which can be constructed using a similar methodology. Interestingly, the decoupling of image point coordinates is still possible, even for the case of perspective projection (assuming that the known views are orthographic [13]). Some results for the orthographic case can be found in [14].

References

[1] 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.

[2] D. Huttenlocher and S. Ullman, "Recognizing solid objects by alignment with an image", *International Journal of Computer Vision*, vol. 5, no. 2, pp. 195-212, 1990.

[3] 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, December 1991.

[4] D. Thompson and J. Mundy, "Three dimensional model matching from an unconstrained viewpoint", *Proceedings of the IEEE Conference on Robotics and Automation*, pp. 208-220, 1987.

[5] E. Pauwels et. al, "Recognition of planar shapes under affine distortion", *International Journal of Computer Vision*, vol. 14, pp. 49-65, 1995.

[6] 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.

[7] G. Bebis, M. Georgiopoulos, N. da Vitoria Lobo, and M. Shah, "Learning affine transformations of the plane for model based object recognition", *13th International Conference on Pattern Recognition*, Vienna, Austria, August 1996.

[8] G. Forsythe, M. Malcolm, and C. Moler, "*Computer methods for mathematical computations*, (chapter 9), Prentice-Hall, 1977.

[9] R. Moore, *Interval analysis*, Prentice-Hall, 1966.

[10] Hertz, A. Krogh, and R. Palmer, *Introduction to the theory of neural computation*, Addison Wesley, 1991.

[11] T. Poggio and S. Edelman, "A network that learns to recognize three-dimensional objects", *Nature*, vol. 343, January 1990.

[12] S. Ullman and R. Basri, "Recognition by linear combination of models", *IEEE Pattern Analysis and Machine Intelligence*, vol. 13, no. 10, pp. 992-1006, October 1991.

[13] A. Shashua, "Algebraic Functions for Recognition", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 8, pp. 779-789, 1995.

[14] G. Bebis, M. Georgiopoulos, and S. Bhatia, "Learning Orthographic Transformations for Object Recognition", *IEEE International Conference on Systems, Man, and Cybernetics*, Vol. 4, pp. 3576-3581, Orlando, FL, October 1997.

[15] G. Bebis, M. Georgiopoulos, M. Shah, and N. La Vitoria Lobo, "Indexing based on algebraic functions of views", accepted for publication, *Computer Vision and Image Understanding (CVIU)*, December 1998.

[16] W. Press et. al, *Numerical recipes in C: the art of scientific programming*, Cambridge University Press, 1990.

[17] A. Neumaier, *Interval methods for systems of equations*, Cambridge Univ. Press, 1990.

[18] E. Hansen and R. Smith, "Interval arithmetic in matrix computations: Part II", *SIAM Journal of Numerical Analysis*, vol. 4, no. 1, 1967.

[19] W. Grimson. D. Huttenlocher, and D. Jacobs, "A study of affine matching with bounded sensor error", *International Journal of Computer Vision*, vol. 13, no 1, pp. 7-32, 1994.

[20] H. Liao, J. Huang, and S. Huang, "Stoke-based handwritten Chinese character recognition using neural networks", *Pattern Recognition Letters*,

[21] Alan M. Fu and Hong Yan, "Object representation based on contour features and recognition by a Hopfield-Amari network", *Neurocomputing*, vol. 16, pp. 127-138, 1997.

[22] J. Feldman, "Efficient regularity-based grouping", *1997 Computer Vision and Pattern Recognition Conference (CVPR'97)*, pp. 288-294, 1997.

[23] P. Havaldar, S. Medioni, and F. Stein, "Extraction of groups for recognition", *Lecture Notes in Computer Science*, vol. 800, pp. 251-261.

Figure Captions

Figure 1. (a) a known view of a planar object (b)-(d) some new, affine transformed, views of the same object generated by considering the affine transformations shown in Table 1.

Figure 2. A pseudo-code description of the sampling procedure for the generation of the training views.

Figure 3. (a) The original neural network scheme, (b) the simplified neural network scheme.

Figure 4. The steps involved in training the SL-NN to approximate the mapping from the space of object's image coordinates to the space of affine transformations.

Figure 5. The procedure used for testing SL-NN's ability to yield accurate predictions.

Figure 6. The test objects used in our experiments along with the corresponding interest points.

Figure 7. The average mean square back-projection error vs ε and the ranges of values observed. The solid line corresponds to the original data while the dashed line corresponds to the PCA transformed data.

Figure 8. The predictions obtained without using the PCA front-end stage (left) and using the PCA front-end stage (right).

Figure 9. The real scenes used to test the performance of the proposed method ((a),(b)) and the interest points extracted ((c),(d)). The back-projection results by using the affine transformation predicted by the SL-NN are also shown ((e),(f)).

Table Captions

Table 1. The affi ne transformations used to generate Figures 1(b)-(d).

Table 2. Ranges of values for the parameters of affi ne transformation.

Table 3. Actual and predicted affi ne transformations.

Table 4. Number of training views and average back-projection mse.

Table 5. Some results illustrating the discrimination power of the networks.

Table 6. Actual and predicted parameters (planar).

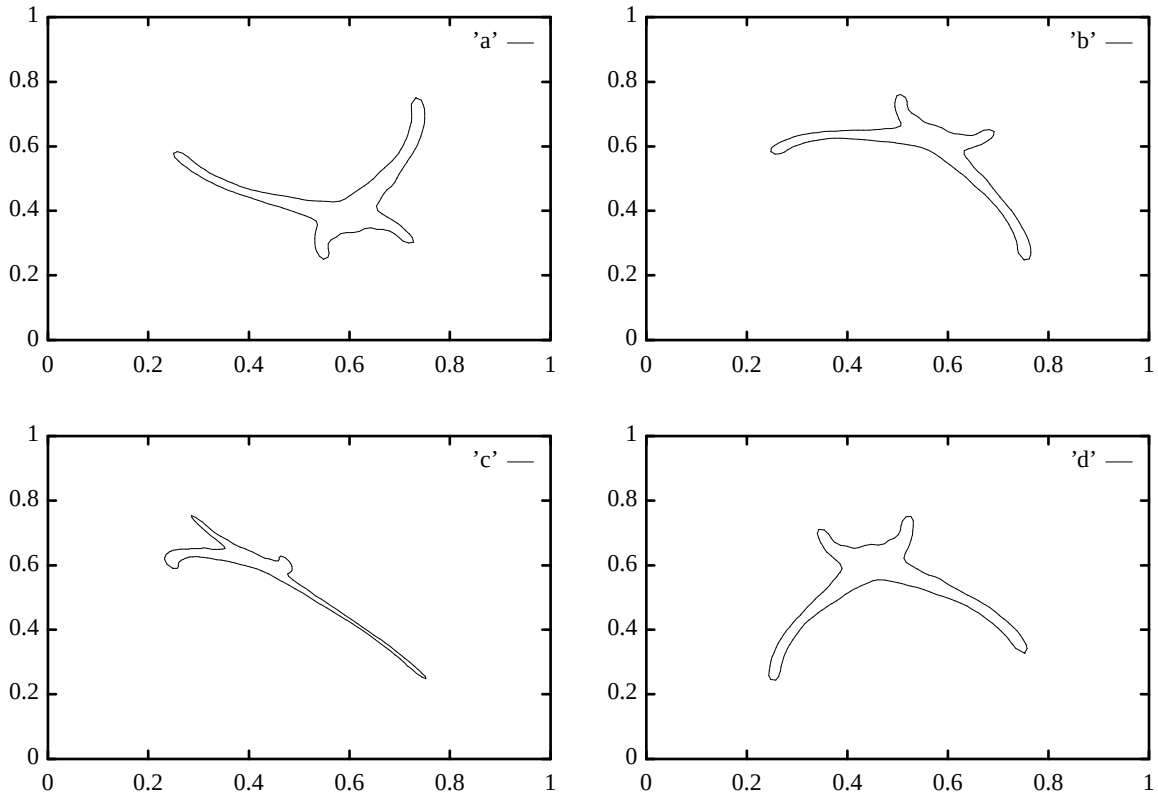


Figure 1

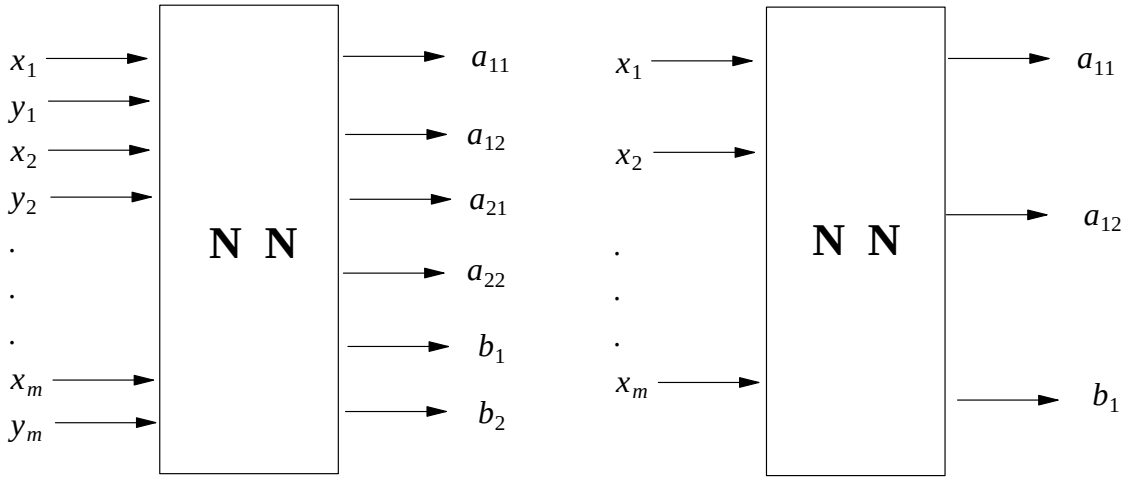
```
for (a11=min_a11; a11 ≤ max_a11; a11 += s_a11)
  for (a12=min_a12; a12 ≤ max_a12; a12 += s_a12)
    for (b1=min_b1; b1 ≤ max_b1; b1 += s_b1)
      for (a21=min_a21; a21 ≤ max_a21; a21 += s_a21)
        for (a22=min_a22; a22 ≤ max_a22; a22 += s_a22)
          for (b2=min_b2; b2 ≤ max_b2; b2 += s_b2) {
              x_i = a11 x_i + a12 y_i + b1
              y_i = a21 x_i + a22 y_i + b2
              if x_i or y_i ∉ [0,1], do not consider
              the current affine transformed view as a training view.
          }
```

(a)

```
for (a11=min_a11; a11 ≤ max_a11; a11 += s_a11)
  for (a12=min_a12; a12 ≤ max_a12; a12 += s_a12)
    for (b1=min_b1; b1 ≤ max_b1; b1 += s_b1) {
        x_i = a11 x_i + a12 y_i + b1
        if x_i ∉ [0,1], do not consider
        the current affine transformed view as a training view.
    }
```

(b)

Figure 2



(a) (b)

Figure 3

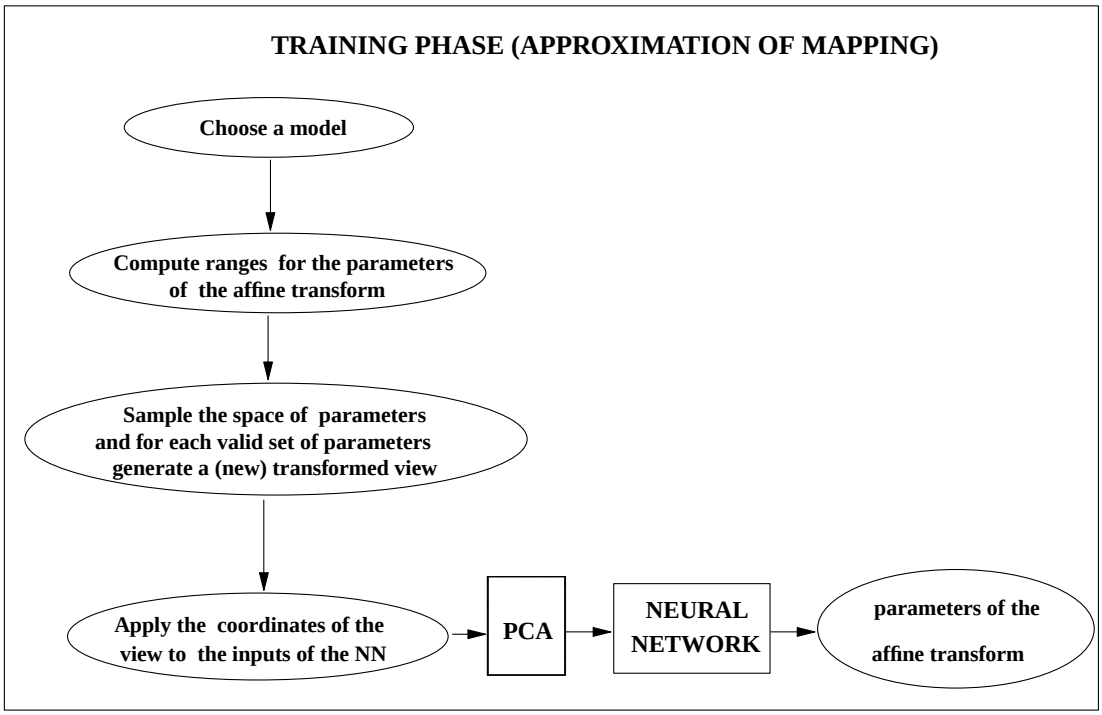

TRAINING PHASE (APPROXIMATION OF MAPPING)
Choose a model
Compute ranges for the parameters of the affine transform
Sample the space of parameters and for each valid set of parameters generate a (new) transformed view
Apply the coordinates of the view to the inputs of the NN
PCA
NEURAL NETWORK
parameters of the affine transform

Figure 4

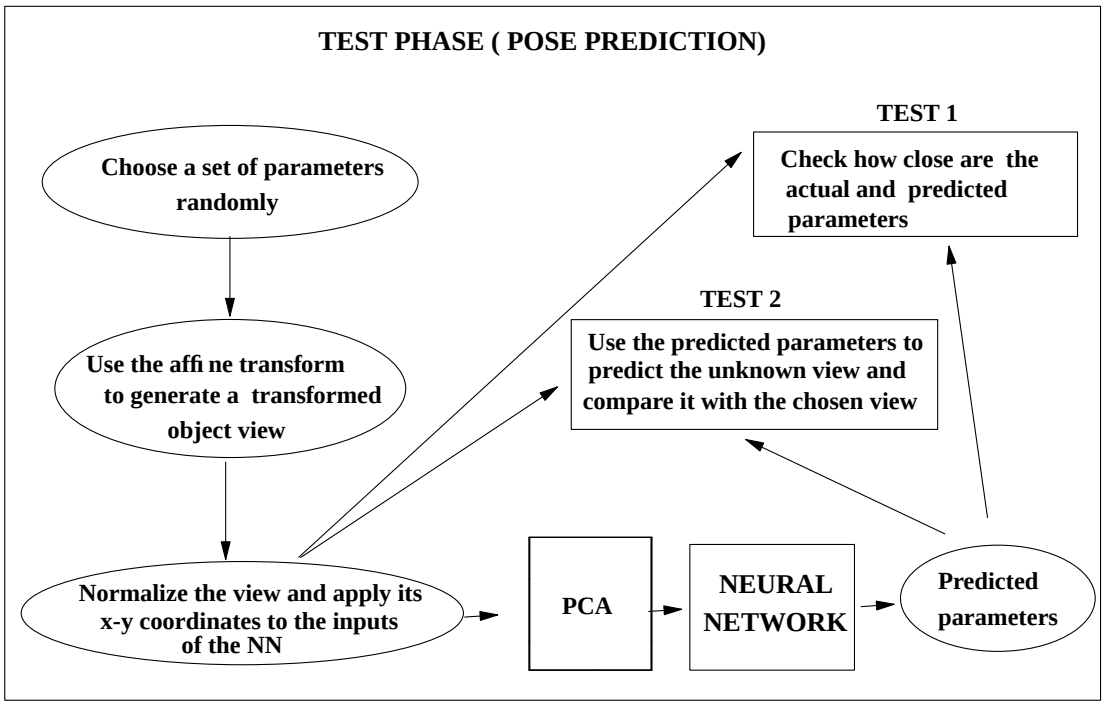

TEST PHASE (POSE PREDICTION)
TEST 1
Check how close are the actual and predicted parameters
Choose a set of parameters randomly
TEST 2
Use the predicted parameters to predict the unknown view and compare it with the chosen view
Use the affi ne transform to generate a transformed object view
Normalize the view and apply its x-y coordinates to the inputs of the NN
PCA
NEURAL NETWORK
Predicted parameters

Figure 5

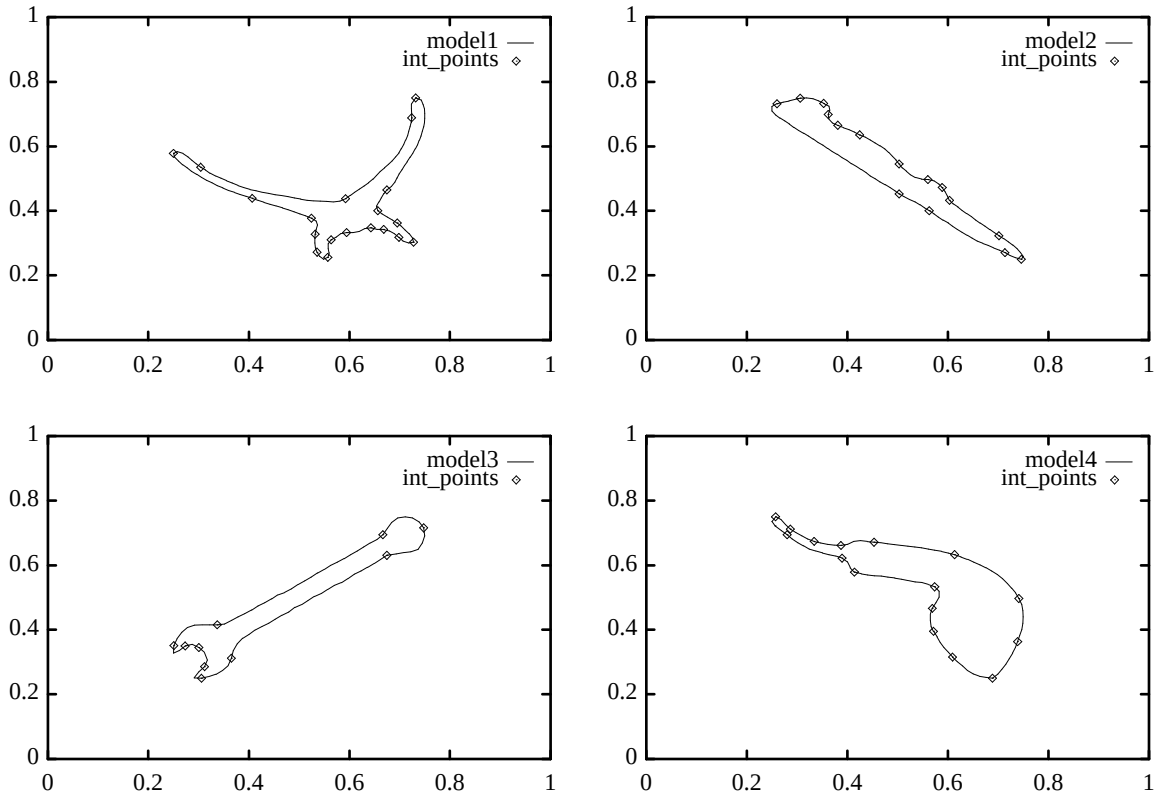

model1
int_points
model2
int_points
model3
int_points
model4
int_points
0
0.2
0.4
0.6
0.8
1

Figure 6

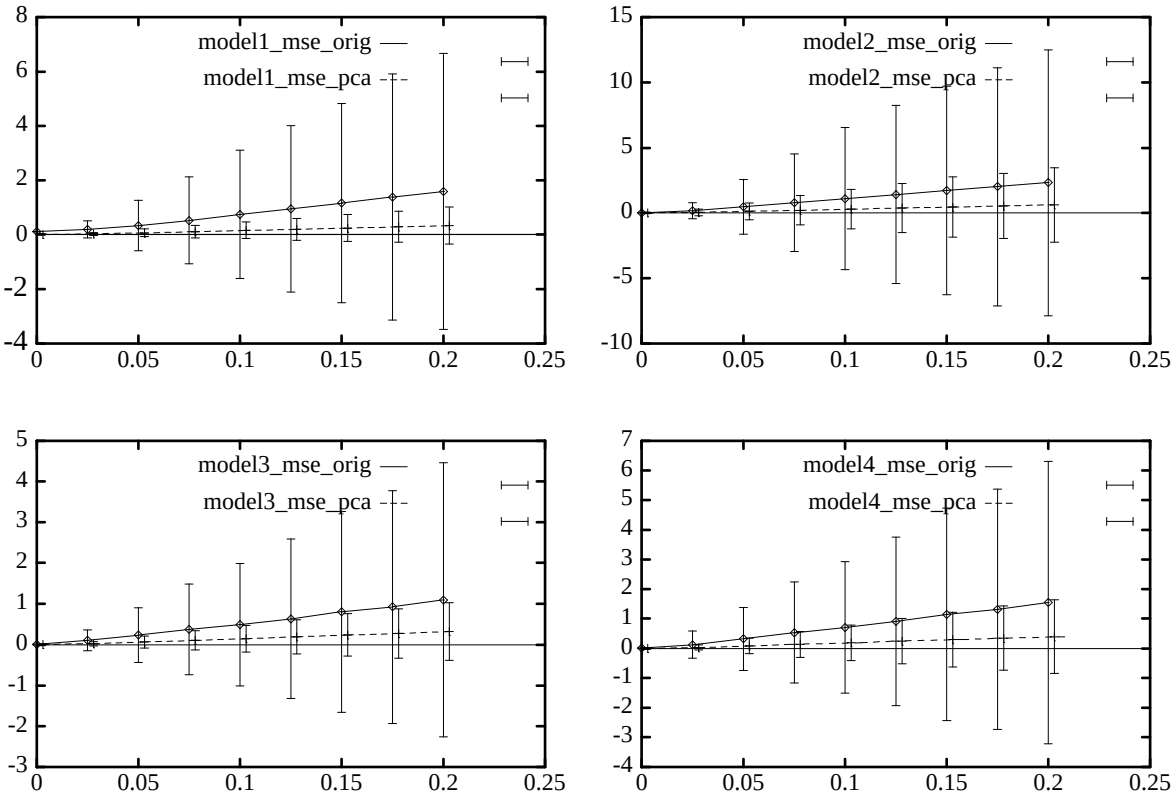

model1_mse_orig
model1_mse_pca
model2_mse_orig
model2_mse_pca
model3_mse_orig
model3_mse_pca
model4_mse_orig
model4_mse_pca

Figure 7

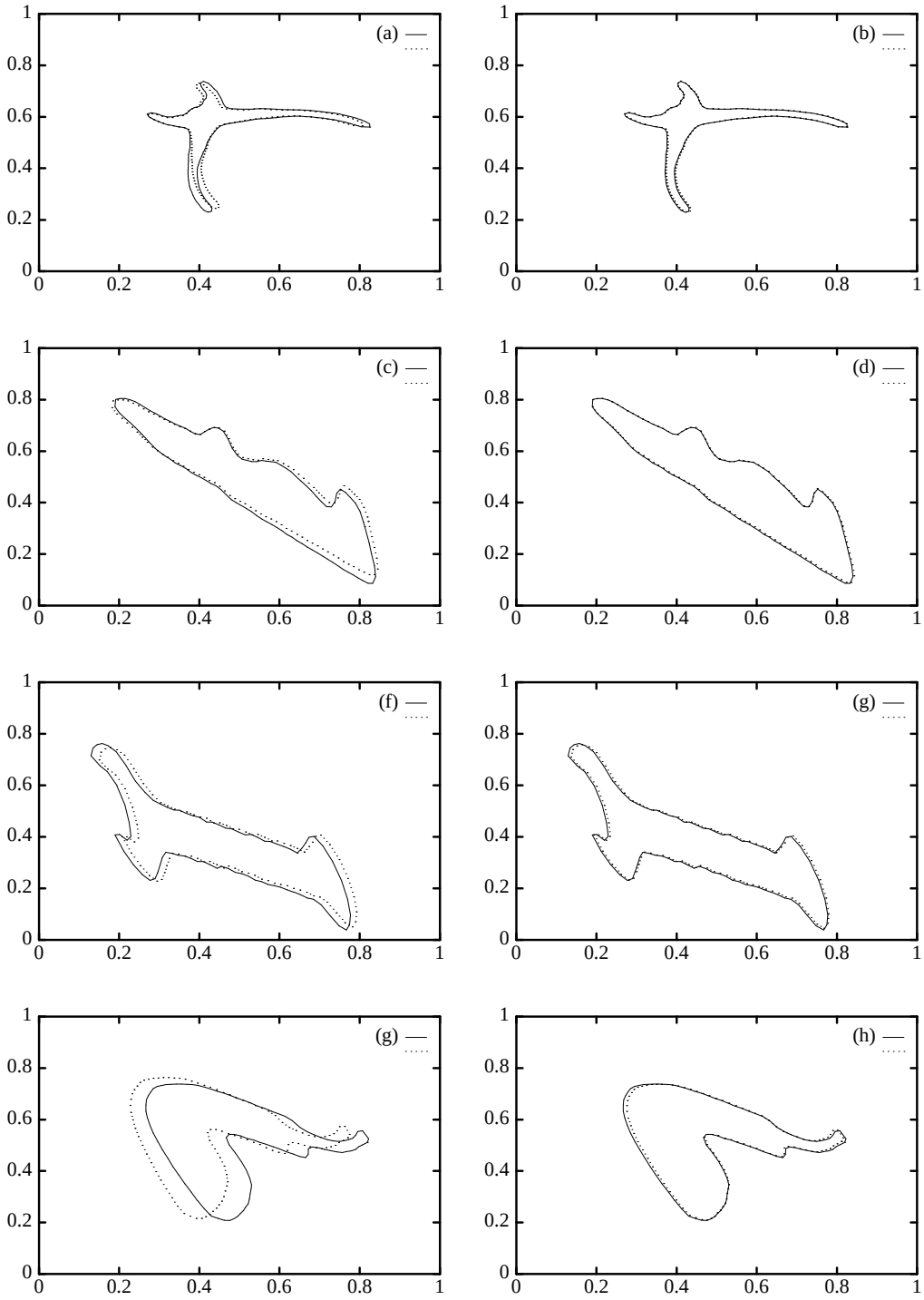
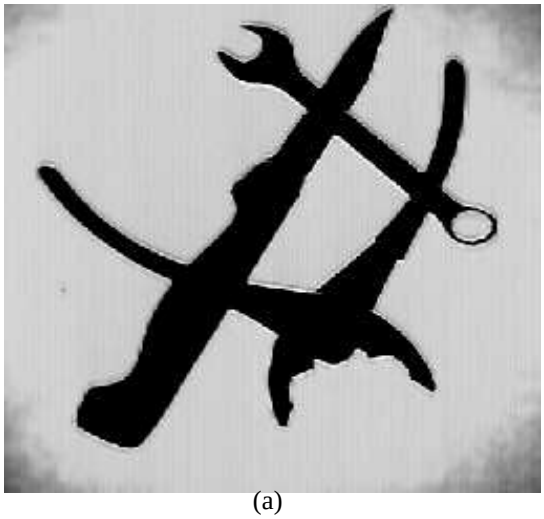
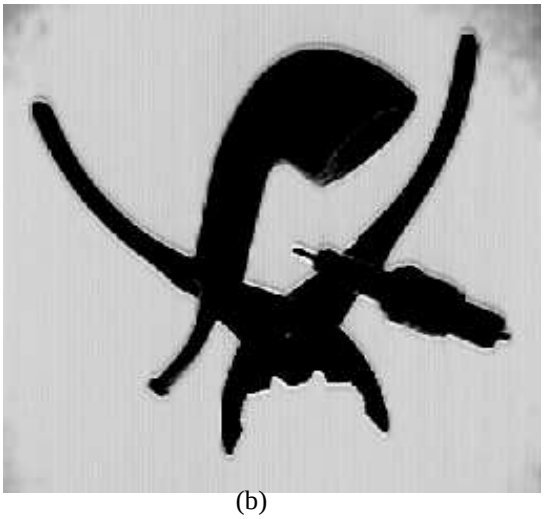


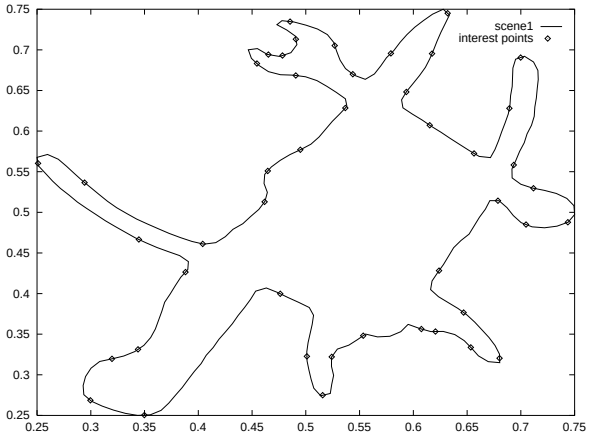
Figure 8



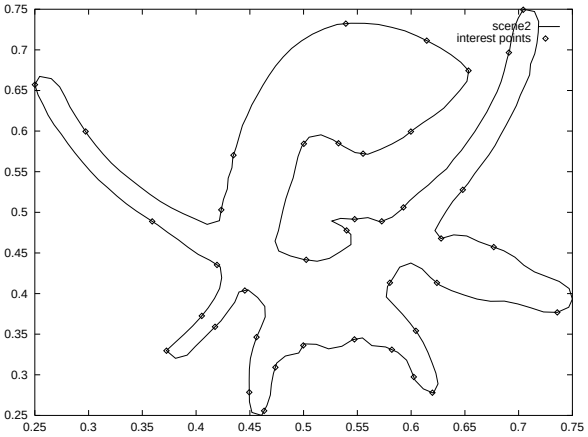
(a)



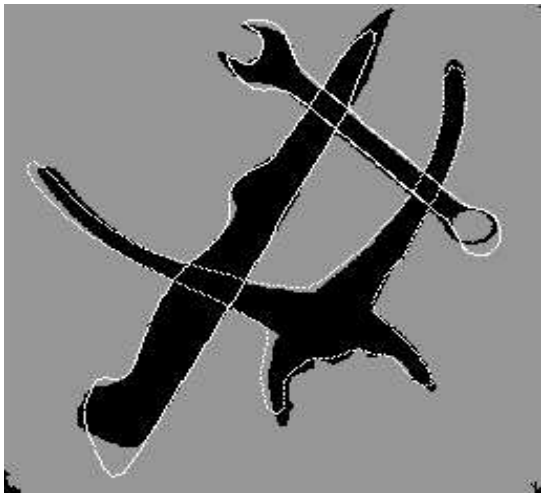
(b)



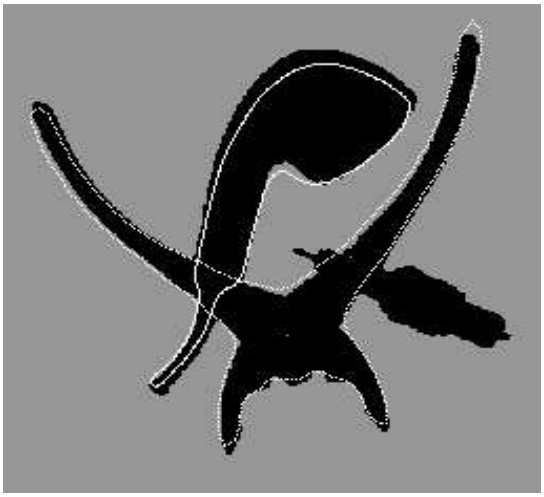
(c)



(d)



(e)



(f)

Figure 9

Parameters of the affi ne transformations			
Parameters	Fig. 1(b)	Fig. 1(c)	Fig. 1(d)
a_{11}, a_{12}, b_1	0.992 0.130 -0.073	-1.010 -0.079 1.048	0.860 0.501 -0.255
a_{21}, a_{22}, b_2	-0.379 -0.878 1.186	0.835 -0.367 0.253	0.502 -0.945 0.671

Table 1

Ranges of values			
	range of a11	range of a12	range of b1
model1	[-2.953, 2.953]	[-2.89, 2.89]	[-1.662, 2.662]
model2	[-12.14, 12.14]	[-11.45, 11.45]	[-11.25, 12.25]
model3	[-8.22, 8.22]	[-8.45, 8.45]	[-0.8, 1.8]
model4	[-4.56, 4.45]	[-4.23, 4.23]	[-4.08, 5.08]

Table 2

Actual parameters			
a_{11},a_{12},b_1	0.6905 -1.4162 0.8265	0.4939 -0.8132 0.7868	-0.3084 -1.1053 1.3546
a_{21},a_{22},b_2	-0.1771 -0.8077 1.2053	0.8935 0.8684 -0.4050	0.2782 -1.2115 1.0551
Predicted parameters (4 training views)			
a_{11},a_{12},b_1	0.6900 -1.4156 0.8265	0.4935 -0.8127 0.7867	-0.3079 -1.1058 1.3537
a_{21},a_{22},b_2	-0.1768 -0.8080 1.2045	0.8921 0.8698 -0.4042	0.2781 -1.2114 1.0547
Predicted parameters (73 training views)			
a_{11},a_{12},b_1	0.6906 -1.4167 0.8269	0.4942 -0.8134 0.7871	-0.3082 -1.1053 1.3550
a_{21},a_{22},b_2	-0.1768 -0.8076 1.2055	0.8938 0.8682 -0.4052	0.2783 -1.2118 1.0554

Table 3

model1					
samples	views	avg-mse	sd	epochs	CPU time (sec)
6-6-6	4	0.122	0.003	7883	4.47
8-8-8	**14**	**0.01**	**0.003**	**20547**	**29.10**
15-15-15	73	0.003	0.001	18736	116.48
model2					
samples	views	avg-mse	sd	epochs	CPU time (sec)
20-20-20	10	49.48	8.1	8876	9.33
26-26-26	**18**	**0.001**	**0.0**	**8798**	**13.83**
30-30-30	32	0.002	0.001	8566	24.87
model3					
samples	views	avg-mse	sd	epochs	CPU time (sec)
6-6-6	6	35.065	6.825	19462	10.38
10-10-10	**14**	**0.006**	**0.002**	**26914**	**29.37**
15-15-15	49	0.005	0.001	23237	75.43
model4					
samples	views	avg-mse	sd	epochs	CPU time (sec)
6-6-6	2	69.392	18.252	6024	1.88
10-10-10	8	0.005	0.001	5774	5.07
14-14-14	**20**	**0.002**	**0.001**	**20262**	**33.20**

Table 4

	model1		model2		model3		model4	
	avg-mse	sd	avg-mse	sd	avg-mse	sd	avg-mse	sd
nn1	0.01	0.003	61.78	21.1	25.6	5.08	51.67	4.42
nn2	292.24	125.31	0.001	0.0	210.21	79.75	187.78	28.06
nn3	114.08	44.96	313.59	79.86	0.006	0.002	48.79	4.88
nn4	110.29	13.35	66.68	20.05	95.77	13.52	0.002	0.001

Table 5

Actual parameters (**Figs. 8(a,b)** and **8(c,d)**)		
a_1,a_2,a_3	-0.063063 -0.120558 0.438347	0.003732 -0.206111 0.530190
b_1,b_2,b_3	0.112543 -0.059775 0.574292	-0.080763 0.152122 0.528324
Predicted parameters (without using PCA)		
a_1,a_2,a_3	-0.066537 -0.112491 0.434431	0.001166 -0.211314 0.530985
b_1,b_2,b_3	0.107072 -0.058731 0.573535	-0.076414 0.145209 0.535657
Predicted parameters (using PCA)		
a_1,a_2,a_3	-0.063082 -0.120546 0.438362	0.003774 -0.206166 0.530237
b_1,b_2,b_3	0.112580 -0.059761 0.574307	-0.080823 0.152155 0.528266
Actual parameters (**Fig. 8(e,f)** and **8(g,h)**)		
a_1,a_2,a_3	-0.227311 0.017821 0.363087	-0.073239 -0.143645 0.570144
b_1,b_2,b_3	0.132470 -0.133993 0.428320	0.116026 -0.063228 0.492276
Predicted parameters (without using PCA)		
a_1,a_2,a_3	-0.226994 0.017980 0.376362	-0.067705 -0.151540 0.520810
b_1,b_2,b_3	0.126998 -0.132556 0.424176	0.121118 -0.063729 0.507291
Predicted parameters (using PCA)		
a_1,a_2,a_3	-0.227329 0.017830 0.363073	-0.073270 -0.143638 0.570132
b_1,b_2,b_3	0.132497 -0.133942 0.428366	0.115976 -0.063170 0.492217

Table 6