Learning Affine Transformations of the Plane for Model-Based Object Recognition

George Bebis†, Michael Georgiopoulos†, Niels da Vitoria Lobo‡, and Mubarak Shah‡

†Department of Electrical & Computer Engineering, University of Central Florida, Orlando, FL 32816
‡Department of Computer Science, University of Central Florida, Orlando, FL 32816

Abstract

In this paper, we consider the problem of learning the mapping between the image coordinates of unknown affine views of an object and the parameters of the affine transformation that can align a known view of the same object with them. A Single Layer Neural Network (SL-NN) is used to learn the mapping. Although the proposed approach is conceptually similar to other approaches in the literature, its practical advantages are more profound. The views used to train the SL-NN are not obtained by taking different pictures of the object but by sampling the space of its affine transformed views. This space is constructed by estimating the range of values that the parameters of affine transformation can assume using a single view and a methodology based on Singular Value Decomposition (SVD) and Interval Arithmetic (IA). The proposed scheme is as accurate as traditional least-squares approaches but faster. A front-end stage to the SL-NN, based on Principal Components Analysis (PCA), increases its noise tolerance dramatically and guides us in deciding how many training views are necessary in order for it to learn a good mapping.

1. Introduction

Affine transformations of the plane or 2-D affine transformations have been widely used in in the area of model-based object recognition [1]-[4]. Given an known and an unknown view of the same planar object, there is an affine transformation that can bring them into alignment. In specific, if p is a point that belongs to the known view and p' is a point that belongs to the unknown view, which are in correspondence, then p' is related to p as follows:

$$p' = Ap + b \qquad (1)$$

where A is a non-singular 2x2 matrix and b is a two-dimensional vector (6 parameters). For any affine view of a planar object, there is a point in the six-dimensional space of affine transformations which corresponds to the transformation that can align the known view with it (in a least-squares sense). In this work, we consider the problem of constructing a function that approximates this mapping. The procedure consists of three main steps. First, we compute the range of values that the parameters of affine transformation can assume. This is performed using Singular Value Decomposition (SVD) [10] and Interval Arithmetic (IA) [5]. Second, we sample the space of affine transformations and for each "sample" affine transform, we use the known view of the object to generate a new affine transformed view. Finally, we train a Single Layer Neural Network (SL-NN) [6] to learn the mapping between the affine transformed views (training views) and the affine transformation which generated them.

Our work has been motivated by [7] and [8]. In [7], the problem of approximating a function that maps any perspective view of a 3-D object to a standard object view was considered. This function was approximated by training a Generalized Radial Basis Functions Neural Network (GRBF-NN). The training views were obtained by sampling the viewing sphere, assuming that the 3-D structure of the object is available. In [8], a linear operator was built which distinguishes between views of a specific object and views of other objects, assuming orthographic projection. This was done by mapping every view of the object to a vector which uniquely identifies the object. Our approach computes the parameters of the transformation that can map the input view to the known view. Obviously, all approaches are conceptually similar. However, our interest here is to benefit methods which operate under the hypothesize-verify paradigm [1],[2]. In this context, it is important to compute the affine transformation as accurately and fast as possible. Accuracy is needed so that verification becomes less ambiguous and speed is important since vast numbers of hypotheses must usually be verified during recognition. We show in section 4.1 that the accuracy of the proposed scheme is as good as applying a traditional least-squares scheme, such as SVD, while its speed is better.

An important advantage of the proposed scheme is that the training views are not obtained by taking different pictures of the object. Instead, they are affine transformed views of the known view which are obtained by sampling the space of affine transformed views which can be constructed using the known view only. On the other hand, the approach in [7] can compute the training views easily only if the structure of the 3-D object is available. Since this is not very realistic, the training views must be obtained by taking different pictures of the object. This, however, requires more effort and time (edges must be extracted, interest point must be detected, and point correspondences across the images must be established). Another advantage is that we do not consider both of the x- and y-coordinates of the object points during training. Instead, we simplify the scheme considerably by decoupling them and by training

The research reported in this paper was partially supported by NSF grant IRI-9220768 and NSF grant CCR-9410459.

the network using only one of the two. Then, during recognition, the parameters of the transformation are predicted in two steps.

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 to the problem of learning to recognize 3-D objects from 2-D views, assuming orthographic or perspective projection. The linear model combinations scheme [8] and the algebraic functions of views [9] can serve as a basis for this extension. In this case, the training views can be obtained by sampling the space of orthographically or perspectively transformed views which can be constructed using a similar methodology. Also, the decoupling of the image point coordinates is still possible, even for the case of perspective projection (assuming that the known views are orthographic [9]).

The organization of the paper is as follows: Section 2 presents the procedure for estimating the range of values for the parameters of the affine transformation. In Section 3, we describe the methodology for obtaining the training views and for training the SL-NN. Our experimental results are given in Section 4. Section 5 follows with our conclusions.

2. Estimating the ranges of parameters

If we assume that each planar object is characterized by a list of "interest" points which may correspond, for example, to curvature extrema or curvature zero-crossings, we can rewrite (1) as follows:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \dots & \dots & \dots \\ x_m & y_m & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ b_1 & b_2 \end{bmatrix} = \begin{bmatrix} x_1' & y_1' \\ x_2' & y_2' \\ \dots & \dots \\ x_m' & y_m' \end{bmatrix} \quad (2)$$

where (x_1, y_1), (x_2, y_2), ... (x_m, y_m) are the coordinates of the points corresponding to the known view and (x_1', y_1'), (x_2', y_2'), ... (x_m', y_m') are the coordinates corresponding to the unknown view (we consider only the points that are common in both views). The above system of equations can be split in two different systems which can be written, using matrix notation, as follows:

$$P_{xy} c_1 = p_{x'} \quad (3)$$

$$P_{xy} c_2 = p_{y'} \quad (4)$$

Both (3) and (4) are overdetermined and can be solved using SVD [10]. SVD produces a solution that is the best approximation in the least-squares sense. Using SVD to factorize P_{xy} we have:

$$P_{xy} = UWV^T \quad (5)$$

where both U and V are orthogonal matrices, while W is a diagonal matrix whose elements w_{ii} are always non-negative (singular values). The solution of the above two systems is $c_1 = P_{xy}^+ p_{x'}$ and $c_2 = P_{xy}^+ p_{y'}$ where P_{xy}^+ is the pseudoinverse of P_{xy} which is $P_{xy}^+ = VW^+U^T$, and W^+ is also a diagonal matrix with elements $1/w_{ii}$ if w_{ii} greater than zero and zero otherwise. The solutions of (3) and (4) are then given by:

$$c_1 = \sum_{i=1}^{3} \left(\frac{U_i p_{x'}}{w_{ii}}\right) V_i \quad (6)$$

$$c_2 = \sum_{i=1}^{3} \left(\frac{U_i p_{y'}}{w_{ii}}\right) V_i \quad (7)$$

where U_i denotes the i-th column of matrix U and V_i denotes the i-th column of matrix V. The sum is restricted over those values of i for which $w_{ii} \neq 0$.

To determine the range of vales for the parameters of affine transformation, we first assume that the image of the unknown view has been scaled so that its x- and y-coordinates belong to a specific interval. This is done by mapping the image of the unknown view to the unit square. In this way, its x- and y-coordinates are mapped in the interval [0, 1]. To determine the range of values for the parameters, we need to consider all the possible solutions of (3) and (4), assuming that the components of the vectors in the right hand side are always restricted to belong in the interval [0,1]. This can be done using Interval Arithmetic (IA) [5]. In IA, each variable is actually represented as an interval of possible values. Given two interval variables $t = [t_1, t_2]$ and $r = [r_1, r_2]$, their sum and product are defined as [5]:

$$t + r = [t_1 + r_1, t_2 + r_2]$$

$$t * r = [min(t_1r_1, t_1r_2, t_2r_1, t_2r_2), max(t_1r_1, t_1r_2, t_2r_1, t_2r_2)]$$

Applying interval arithmetic operators to (6) and (7) instead of standard arithmetic operators, we can compute interval solutions for c_1 and c_2 by setting $p_{x'}$=[0,1] and $p_{y'}$=[0,1]. In interval notation, we want to solve the systems $P_{xy}c_1 = p_{x'}^I$ and $P_{xy}c_2 = p_{y'}^I$, where the superscript I denotes an interval vector. The solutions c_1^I and c_2^I should be understood to mean $c_1^I = [c_1\colon P_{xy}c_1 = p_{x'},\ p_{x'} \in p_{x'}^I]$ and $c_2^I = [c_2\colon P_{xy}c_2 = p_{y'},\ p_{y'} \in p_{y'}^I]$. Since both interval systems involve the same matrix P_{xy} and $p_{x'}$, $p_{y'}$ assume values in the same interval, the solutions c_1^I and c_2^I will be the same. Thus, we consider only the first of the interval systems in our analysis.

By merely applying the interval arithmetic operators to (6) we will most likely obtain a non-sharp interval solution [11]. An interval solution is considered non-sharp if it includes many solutions which do not satisfy the problem at hand (invalid solutions) [11]. Sharp interval solutions are desirable in our approach because they can save us time during the generation of the training views (see next section). One well known factor that affects sharpness is when an interval variable enters the computation of the same quantity more than once [11]. This is actually the case with (6). To make it clear, let us consider the solution for the $i-th$ component of c_1, $1 \le i \le 3$:

$$c_{i1} = \sum_{k=1}^{m} \frac{V_{ik}}{w_{kk}} \left(\sum_{j=1}^{m} U_{jk} x_j'\right) \quad (8)$$

Clearly, each x_j' $(1 \le j \le m)$ enters in the computation of c_i

more than once. To avoid this, we factor out the x_j' and apply the interval arithmetic operators to the next equation:

$$c_{i1} = \sum_{j=1}^{m} x_j' \left(\sum_{k=1}^{m} \frac{V_{ik} U_{jk}}{w_{kk}} \right) \quad (9)$$

3. Learning the mapping

First, we generate the training views by sampling the range of values that the parameters of affine transformation can assume. The sampling procedure is straightforward: we pick a sampling step and we sample the range of values associated with each parameter. For each parameter, we pick one of its sampled values and we form a set of sampled parameter values. This set defines an affine transformation which is applied on the known view to generate a new affine transformed view.

```
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 1. *Generation of the training views.*

Although invalid solutions have been reduced, they might not have been eliminated completely. Consequently, 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 $p_{y'}^I$ [11]. In other words, if we generate affine transformed views by choosing the parameters of affine transformation from the interval solutions computed, then not all of the generated views (actually their interest points) will lie in the unit square completely. These views are invalid and must be disregarded as shown in Figure 1a.

It is important to notice now that since the ranges for (a_{11}, a_{12}, b_1) are the same with the ranges for (a_{21}, a_{22}, b_2), the information generated for x_i' and y_i' will be the same. Since it is redundant to generate the same amount of information twice, we generate information only about the x-coordinates (see Figure 1b). In this way, the time and space requirements of the scheme are significantly reduced. Furthermore, a network of half the size is needed (see Figure 2b) which implies faster training. The only additional cost due to this simplification is that the parameters of the transformation must now be predicted in two steps: first, we must present to the network the x-coordinates of the unknown view to predict (a_{11}, a_{12}, b_1) and then we must present the y-coordinates to predict (a_{21}, a_{22}, b_2).

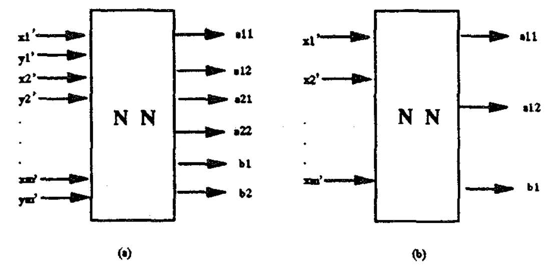

Figure 2. *(a) The neural network scheme, (b) the simplified neural network scheme.*

4. Experiments

4.1. Evaluation of SL-NN's performance

Figure 3 shows the four different objects used in our experiments and the "interest" boundary points extracted (curvature extrema and zero-crossings). The computed ranges of values for the parameters of affine transformation are shown in Table 1. For each object, we generated a number of training views and we trained a SL-NN to learn the desired mapping. Back-propagation with momentum was used [6]. The learning rate used was 0.2 and the momentum term was 0.4. The network assumed to have converged when the sum of squared errors between the desired and actual outputs was less than 0.0001.

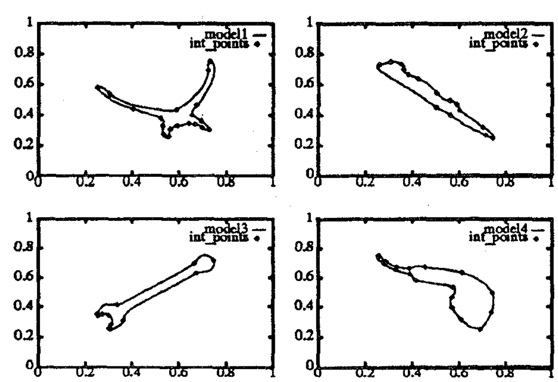

Figure 3. *The test objects used.*

To evaluate the quality of the mapping computed by the SL-NN, we generated a number of test views per object, by affinely transforming the known views choosing the transformation parameters randomly. To ensure that the x- and y-coordinates of the test views belong in [0,1], we chose a random subsquare within the unit square and we mapped the square enclosing the view of the object to the randomly chosen subsquare. To find how accurate the predictions made by the SL-NN are, we compared the parameters of the predicted transformation with the parameters of the actual transformation which we computed using SVD. Also, we back-projected the known view on the test view and we computed the mean-square error between the two [1],[2]. Table 2 shows some affine transformations predicted by a network trained with only 4 views in the case of model1. These views were generated by sampling each parameter's range at 6 points. Invalid views were not included in the training set (see section 3). The actual affine transformations are also shown for comparison. We also show results using 73 training views which were generated by sampling each parameter's range at 15 points.

Table 1. *Ranges for the parameters.*

Ranges of values				
	interest points	range of a11	range of a12	range of b1
model1	19	[-2.953, 2.953]	[-2.89, 2.89]	[-1.662, 2.662]
model2	15	[-12.14, 12.14]	[-11.45, 11.45]	[-11.25, 12.25]
model3	10	[-8.22, 8.22]	[-8.45, 8.45]	[-0.8, 1.8]
model4	16	[-4.56, 4.45]	[-4.23, 4.23]	[-4.08, 5.08]

Table 2. *Actual and predicted affine transformations.*

Actual affine transformations			
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 affine transformations (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 affine transformations (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 demonstrates the performance of the SL-NN, using various numbers of training views. For each case, we report the number of points at which each parameter's range was sampled to generate the specified number of training views, the average mean square back-projection error, the standard deviation of error, and the training time (epochs, CPU time). The error was computed using 100 test views for each object. The results indicate that the SL-NN is capable of approximating the desired mapping very accurately, using a small number of training views (4-15 in our experiments).

We also examined the computational requirements of the SL-NN, assuming that training is done off-line. If m is the average number of interest points per model and n the number of parameters (n=6), the SL-NN requires nm multiplications and nm additions to compute the parameters of the transformation. In the case of the traditional least-squares approach we used here to compute the actual values of the parameters (SVD [10], p. 65), we need $n(m+n)$ multiplications, nm divisions, and $n(m+n)$ additions, assuming that the factorization of P_{xy} has been done off-line. Since these computations are repeated hundreds of times during recognition, the neural network approach is obviously superior.

Table 3. *Number of training views and average mse.*

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**

4.2. Discrimination power

The term "discrimination power" means the capability of a SL-NN to predict a wrong affine transformation if it is shown a view belonging to an object which is different form the object whose views were used to train it (object specific networks). For each object, we used the SL-NN trained with the number of training views shown highlighted in Table 2. Since each 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 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, based on PCA [10], for reducing the dimensionality of the input data first. In this way, all the networks will have exactly the same number of input nodes. PCA might have additional benefits for the performance of the networks because the new inputs are uncorrelated which implies faster training and probably better generalization. Table 4 illustrates the results (100 test views per model were used).

Table 4. *Discrimination power of the networks.*

	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

4.3. Noise and occlusion tolerance

To test the noise tolerance if the method, we assume that the location of each interest point can be anywhere within a disc centered at the real location of the point and having a radius equal to ε. To test the networks, we used a set of 100 test 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 4, solid lined). More training views did not improve the results significantly.

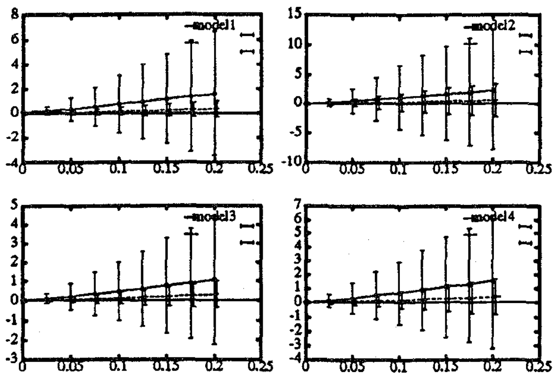

Figure 4. *The average mse vs ε.*

Then, we tested the performance of the method assuming that the front-end stage is active now. What we observed is quite interesting. In cases where the performance of the method was poor, we found that the number of non-zero eigenvalues associated with the covariance matrix of the training views was consistently less than three. More training views did not improve the results, as long as the number of non-zero eigenvalues remained less than three. Including enough training views so that the number of non-zero eigenvalues became three, resulted in a dramatic error decrease. Even more training views did not help significantly and the number of non-zero eigenvalues remained three. The same observations were made for all the four objects we used. We believe that the reason there are three non-zero eigenvalues is related to the three unknown parameters (for the x-coordinates) of the mapping we approximate. Since the training views we pick might not always be representative of the space of affine transformed views, PCA can guide us in choosing a sufficient number of training views so that the network can compute a good, noise tolerant, mapping.

Assuming some of the interest points to be occluded and the front-end stage to be inactive, resulted in a very poor performance, even with one point missing. When the front-end stage was activated, an improved performance was observed but only when 2-3 points were missing at most. This suggests that in order to deal with occlusion, it is more appropriate to use groups of points in training.

4.4. Performance using real scenes

Here, we considered the real scenes shown in Figure 5. Point correspondences were established by hand. When a model point did not have an exact corresponding scene point, we chose the closest possible scene point. Also, when a model point did not have a corresponding scene point because of occlusion we just picked the point (0.5, 0.5) (the center of the unit square) to be the corresponding scene point. For all the models present in the scenes, the transformation computed was quite good. Figure 5 illustrates the results.

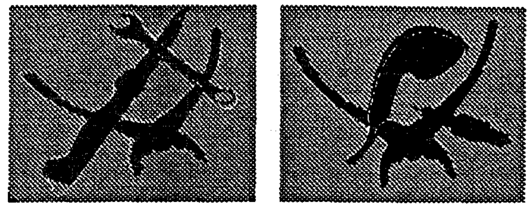

Figure 5. *The real scenes used.*

5. Conclusions

We considered the problem of learning the mapping from the space of object image coordinates to the space of affine transformations. The proposed approach has more practical benefits than similar approaches. Extensions to the recognition of 3-D objects are currently being explored.

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

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

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

[8] 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.

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

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

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