University of Tehran
Faculty of Engineering
Electrical and Computer Engineering Department
Pattern Recognition Project

Two-dimensional Shape Classification

Implementation and Evaluation in Noise-free and Noisy Outlier Contaminated Environments

FARSHAD SHIRANI

FALL – 2003

Abstract

In this project we consider the problem of classifying objects using their two-dimensional silhouettes in noise-free environments as well as environments generating large aberrant observations (outliers). We concentrate on fast classification algorithms for near real-time applications. In order to obtain acceptable results in both environments, two independent classifying methods are presented and examined through three sets of shape. Both these methods are generally based on the use of circular autoregressive (CAR) model parameters which represent the shape of the boundaries detected in digitized binary images of the objects. Robust parameter estimation and lag selection procedures are introduced and used in contaminated environment. All object identification techniques are insensitive to object size and orientation.

All techniques and algorithms are implemented with MATLAB and contaminations are generated artificially.

1. Introduction

Recognizing two or three dimensional objects is a central problem in machine vision, with applications to aircraft identification, medical diagnosis from cell characteristics, hand written character recognition, automatic inspection of industrial processes, and so on. We focus on techniques based on two dimensional boundary information of shapes whose boundary does not cross itself. Shape information is obtained by the use of a circular autoregressive (CAR) model representing the objects boundary.

Section 2 describes the boundary modeling used in our all algorithms. In section 3 the classification problem in noise-free environment is considered. The feature extraction procedure based on model parameter estimates is carried out in section 3.1. Two fast classifiers are introduced in section 3.2 and their performance is tested in section 3.3. Section 3.4 examines the ability of our approach in recognizing deformed shapes.

Classification in contaminated environment is the subject of section 4. The model is introduced in section 4.1. Robust parameter estimation and lag selection algorithms are presented in sections 4.2 and 4.3. A new classification approach based on spectral functions estimated from model parameters is explained in section 4.4. In section 4.5 we indicate the insensitivity of robust methods to outliers through some spectral function diagrams. The classification algorithms are also tested in this section. Finally, we summarize the conclusions in section 5.

2. The Boundary Model

Our shape description technique is based on the use of a full circular autoregressive (CAR) model of order *M* in noise-free environment and a circular subset autoregressive (CSAR) model in contaminated environment. An *autoregressive model* is a parametric equation that expresses each sample of an ordered set of data samples as a linear combination of a specified number of previous samples from the set plus an error term.

The autoregressive model can be used to express a polygonal approximation of a two-dimensional object boundary. With appropriate boundary sampling, functions of the model parameters are invariant to rotation, translation, and scaling of the boundary and can be used as shape descriptors.

In order to obtain CAR model parameters of a given shape, we must first extract its boundary. Then the boundary is approximated by a sequence of ordered samples and represented by a set of CAR model parameters. These procedures are what we are going to explain in sections 2.1-2.3.

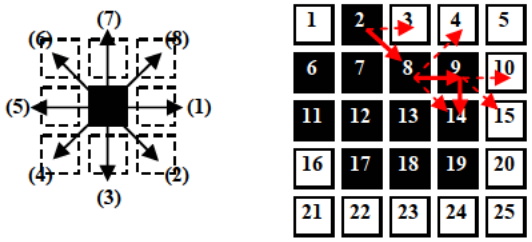
2.1. Boundary Extraction

Suppose we have an isolated black and white image of an object (e.g. black object in a white background). Driving this image from the original image taken by camera might need some preprocessing that is not the subject of this project. In order to extract the boundary of this image we suggest this simple algorithm

1) Find the topmost pixel of the object and store it as a boundary point. Also choose this point as starting point.
2) Define eight different directions to access all neighbor pixels of current point and order them in clockwise direction starting at left direction (see Fig. 1). Choose direction (1) as current direction.
3) Check the current point neighbor pixel in current direction.
   If it does not belong to the object, check the pixel in next direction. Repeat this until you reach to a pixel which belongs to the object. Store this pixel as a boundary point and choose its access direction as current direction. Also choose this point as current point.
   Else if it belongs to the object, check the pixel in previous direction. Repeat this until you reach to a pixel not in the object. Store the last detected object pixel as a boundary pixel and choose its access direction as current direction. Also choose this point as current point.
4) If the current point is not the starting point, go back to step 3), other wise delete this last stored point and terminate the program.

This algorithm is shown graphically for a few steps in Fig. 1. As we can see in this figure, pixels around the current pixel are checked one by one until an Off-to-On or On-to-Off transition occurs in the state of

pixels. This transition indicates a pass through the object boundary.



Current point	Checking direction	Pixel being checked	Pixel state	Action done	Boundary Points
2	---------	----------	----	Store starting point	{2}
2	(1)	3	Off	Check pixel in next direction	{2}
2	(2)	8	On	Store this point	{2,8}
8	(2)	14	On	Check pixel in previous direction	{2,8}
8	(1)	9	On	Check pixel in previous direction	{2,8}
8	(8)	4	Off	Store previous point	{2,8,9}
9	(1)	10	Off	Check pixel in next direction	{2,8,9}
9	(2)	15	Off	Check pixel in next direction	{2,8,9}
9	(3)	14	On	Store this point	{2,8,9,14}
14	.	.	.	. . .	. . .
.	.	.	.	. . .	. . .
.	.	.	.	. . .	. . .
.	.	.	.	. . .	. . .

Fig. 1. Few steps of the boundary extraction procedure

Performing morphological 'removing' operation on our black and white image or applying some edge detection algorithms will also leave the object boundary. These operations are provided in MATLAB and in some cases might have better results than our algorithm. However, because we need to have an ordered sequence of boundary points, we must do some further processing on the boundary detected by these functions to generate our desired boundary sequence. This extra processing will be almost similar to our own algorithm that detects the boundary and generate its points sequence simultaneously. Therefore we prefer to use our own simple algorithm.

We might encounter some problems when we use this algorithm for objects including very sharp (needle shaped) parts on their boundary. These parts might not be detected by our algorithm, so if the stating point lies on one of these parts the procedure won't terminate. Another problem can be made by objects which have some holes so near to their boundary that there is just one pixel separating inside the hole and outside the boundary. In such cases, following our algorithm, we might lose the object boundary and enter the hole. The detected boundary will then circle itself round the hole and the procedure won't terminate again. In practice such holes can be generated by noise in shapes containing two parts too close to each other (deep narrow concavities in the object). In the presence of noise, these parts can be easily connected to each other and leave a hole inside the object. As shown in Fig. 2 this problem is encountered in 'Goose' (see aircraft templates in Fig. 8) when it is rotated (or scaled) over particular degrees.

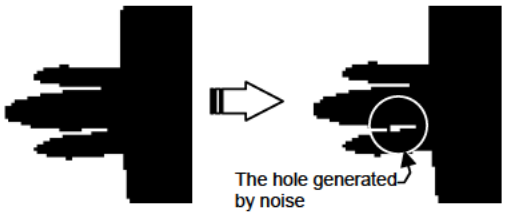


Fig. 2. Noise generated holes

In order to prevent such undesirable effects, we perform a special pre-filtering on images before starting the boundary extraction process. This filtering task includes two steps. First we remove all those object pixels which have less than or equal to three pixels in their neighborhood. This step almost eliminates all troublesome sharp parts. Then we perform a morphological 'closing' operation ('dilation' followed by 'erosion') on the image obtained from previous step. In this step, the above-mentioned noise generated holes are filled completely if they are narrow enough, or at least the separating part between them and outside the object is widened. In both these cases, holes are not detected at all by our algorithm and can not make any problem.

2.2. Boundary Representation [2]

In order to construct a shape model, model parameters are estimated from a data set of boundary samples. The boundary extracted in previous section is approximated by an ordered sequence of the lengths of N angularly equispaced radius vectors projected between the boundary centroid and the boundary, as shown in Fig. 3.

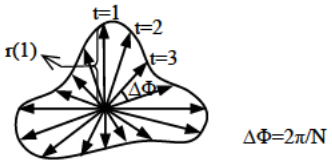


Fig. 3. Boundary approximation

This boundary approximation can be improved by increasing the number of radius vector projections N. The radius vector lengths r are a function of the

angle of projection $\Phi = t2\pi/N$, where $t = 1, 2, \ldots, N$, and the function $r(\Phi)$ forms a one-dimensional boundary approximation. As Fig. 4 shows this ordered set of numbers, r, can also be expressed as a "time series" $r(t)$, with the parameter t describing the position of a radius vector in equiangular increments from the starting point. It is clear that for closed boundaries this is a periodic time series.

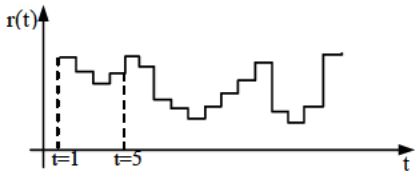


Fig. 4. Plot of $r(t)$ versus t for the shape in Fig.

By considering a more complicated boundary than the one shown in Fig. 3 an initial problem of this sampling method is clarified. For non-convex boundaries the time series described above becomes a multi-valued function. Dubois and Glanz [2] developed an algorithm that "unwraps" the boundary and "stretches" the multi-valued function to produce a one-dimensional function $r(t)$. This scheme searches sequentially a long the boundary until a radius vector crossing is detected and measures the length. To determine the next ordered length, the sequential boundary search is continued until another radius vector crossing is detected. Thus the radius vector lengths are ordered according to the order of detection t. The sequence of boundary points was generated in section 2.2, so here, we check boundary points one by one to detect radius vector crossings and store successively the lengths of vectors. This new scheme also produces a periodic or circular time series but obviously with a longer period T. Fig. 5 shows this procedure.

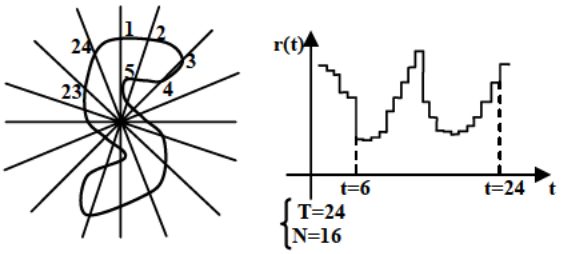


Fig. 5. Concave shape radius vectors and its unwrapped time series

The practical application of this scheme to non-convex shapes requires some certain assumptions that are described below.

If N radius vectors are projected between the boundary centroid and the boundary, this scheme always detects more than N radius vector-boundary intersections for a non-convex shape. The number of intersections varies slightly for identically shaped objects in different rotational positions. If N is large enough, the variation of the number of intersections is small compared to the total number of intersections and does not affect the model invariability to rotation.

If the boundary of an object has segments that are straight lines, certain rotations of the object may cause many consecutive points of a line segment to intersect single radius vector, as shown in Fig. 6. If all of the lengths along a radius vector to boundary points of such a line segment are included in the time series, the least square fit of the CAR model to the time series will be biased. Also a slight rotation of the shape results in the entire line segment points being undetected, and the model parameters are estimated from a much smaller number of observations.

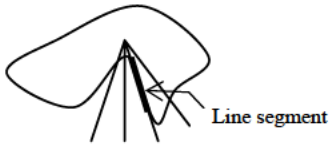


Fig. 6. Example of collinear radius vector and boundary segment

To prevent this situation, we allow only one radius vector-boundary intersection between consecutive boundary points and a particular radius vector.

The algorithm proposed in [2] for implementing this scheme is described here (with some changes) through the following steps.

1) Choose the number of radius vectors N to project from the boundary centroid. This number is assumed to be an integral multiple of four $(N = 4k, k \in \mathrm{N})$ to simplify the algorithm in next step.
2) Calculate and store the magnitude of the radius vector slopes for the first quadrant. Because we use an ordered sequence of boundary points to obtain an ordered set of radius vector-boundary intersection lengths, we need not determine the actual quadrant in which a radius vector lies. Since N is assumed to be an integral multiple of four, the magnitudes of slopes of radius vectors in all other quadrants are equal to those of the first quadrant.
3) Begin the search at the starting boundary point $P_1 = (x_1, y_1)$.
4) Compare this point with the centroid $C = (x_c, y_c)$. If $x_1$equals x_c, the starting point is an intersection point with a vertical radius vector, or if $y_1$equals y_c, the starting point is an intersection point with a horizontal radius vector. If any of these cases occurs, skip steps 5) - 7) and jump to step 8), otherwise proceed to step 5).
5) Calculate the slope magnitude of the line containing the centroid and the starting point as follows:

$$slope_{p_1} = \left| \frac{y_1 - y_c}{x_1 - x_c} \right|$$

From the array of slopes find the two slopes between which the calculated slope lies. Suppose they are Slope(i) and Slope(i+1) of the radius vectors i and i+1 (see Fig. 7). These two slopes define a sector.

6) Calculate the differences, $RSD1$ and $RSD2$, between $slope_{p1}$ and two sector slopes, Slope(i) and Slope(i+1):

$$RSD1 = slope_{p_1} - Slope(i)$$
$$RSD2 = slope_{p_1} - Slope(i+1)$$

These values are references for the sector and the point P_1 is called the sector reference point.

7) Choose the next boundary point. Let it in general be the point P_k. Check step 4) for this point. If the conditions in step 4) does not apply to this point continue, otherwise do what step 4) says.
Calculate the differences $SD1$ and $SD2$ for this point:

$$slope_{p_k} = \left| \frac{y_k - y_c}{x_k - x_c} \right|$$
$$SD1 = slope_{p_k} - Slope(i)$$
$$SD2 = slope_{p_k} - Slope(i+1)$$

Compare $SD1$ with $RSD1$, and $SD2$ with $RSD2$. If $SD1$ and $RSD1$ have opposite signs, P_k is the approximate radius vector i boundary intersection point. If $SD2$ and $RSD2$ have opposite signs, as in Fig. 7, P_k is the approximate radius vector i+1 boundary intersection point. If there is no sign change repeat this step.

8) If the intersected radius vector is not the same as the radius vector of the previous intersection, calculate the radius vector length and store it as a function of the order in which it was detected $r(t)$, otherwise, discard the point with out further processing.

9) Choose the next boundary point that is actually the first point of the new sector. Go back to step 4) and repeat the procedure interpreting this point as the starting point P_1 in all steps. Of course if there is no boundary point left, the algorithm terminates.

Some points about this algorithm must be noted here.

a. As the algorithm implies, we first check every boundary points to see whether they are on vertical or horizontal radius vectors or not. The vertical vectors are checked because their slopes are infinity and can not enter into our calculation. The purpose of checking horizontal vectors is that the boundary points intersected by these vectors can not be detected through steps 5) – 7). This is a consequence of our one quadrant scheme. In other words, because we consider only the magnitude of slopes and not also their sign, the algorithm does not sense any changes when it passes across the horizontal radius vectors in which only the sign of vector slopes changes.

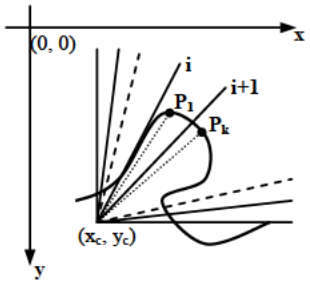


Fig. 7. Radius vector-boundary intersection detection example

b. Since the boundary is supposed to be connected, when an intersection is detected, one might say we do not need to locate the new sector again by searching among the array of slopes (except for the starting boundary point). It means that because the sectors are traced consecutively, we do not need to execute step 4) each time we come back from step 9) and the new sector is known according to the previous sector and the detected radius vector-boundary intersection. This is true if we forget we are dealing with a digital image! Because of the digitalization effect, or in other words because we have a finite number of boundary points, the boundary is not really a connected path. This unconnected nature of the boundary emerges where the boundary goes too close to the centroid (see 'Boomerang' in Fig. 9). In this case, several radius vectors can lie between two adjacent boundary points and a jump across sectors will occur when we pass a long these points. Thus we do need to execute step 4) to determine the new sector we entered into, each time an intersection is detected.

c. As explained above, one boundary point near the centroid can mathematically intersect more than one radius vector. In our algorithm we automatically save only the first detected intersection.

d. Because a change in sectors can happen between the last boundary point and the first one (interpreting circularly), we add the first (starting) boundary point to the end of the sequence of boundary points if this point is not on a horizontal or vertical radius vector. In this way we are able to detect the above-mentioned intersection too.

We obtain our desired boundary representing time series by implementing the algorithm described in this section and are able to construct the shape autoregressive model. This is the subject which is discussed in next section.

2.3. Mathematical modeling of the boundary [3]

In this section we consider the analyses of the one-dimensional time series {$r(1)$, $r(2)$, . . . , $r(T)$} derived from the methods discussed in previous sections. Since the boundary is closed,

$$r(k+T) = r(k)\ ,\ \forall\ \text{int}\,eger\ k \qquad (1)$$

we fit a particular type of CAR model to this data (see [3]):

$$r_t = \alpha + \sum_{j=1}^{m} \theta_j\ r_{[t-t_j]} + \sqrt{\beta}\ w_t \quad t = 1, \cdots, T \qquad (2)$$

where

r_t = current radius vector length

$r_{[t-t_j]}$ = the radius vector length detected t_j radius vectors before the current r_t (collectively called lag terms). Here [x] is x interpreted periodically on the integers 1, 2, …, N

$\theta_1, \cdots, \theta_m$ = unknown lag coefficients to be estimated from the observed time series

m = model order

$\sqrt{\beta}$ = unknown constant to be estimated

$\sqrt{\beta}\ w_t$ = current error, noise, residual

α = unknown constant to be estimated

$\{w_t\}$ = a random sequence of independent zero-mean samples with unit variance: $E(w_i) = 0$, $E(w_i w_j) = \delta_{ij}$

Since the variance of w_t is one, the $\sqrt{\beta}$ factor transforms the unit variance random variable w_t to a random variable with variance β.

The unknown model parameters $(\alpha, \boldsymbol{\theta}, \beta)$ in Eq. (2) are estimated from the observed time series. The popular estimation of the parameters is the ML estimation. If we let $(\alpha^*, \boldsymbol{\theta}^*, \beta^{*)}$ be the ML estimation of the parameters $(\alpha, \boldsymbol{\theta}, \beta)$, then $\boldsymbol{\theta}^*$ can be computed by a gradient algorithm. For reducing the computations in the estimation procedure, the use of the following LS estimation is suggested by [3].

$$(\hat{\boldsymbol{\theta}}^T, \hat{\alpha}) = \left[\sum_{t=1}^{T} \boldsymbol{u}_{t-1}\boldsymbol{u}_{t-1}^T\right]^{-1}\left[\sum_{t=1}^{T} \boldsymbol{u}_{t-1} r_t\right], \qquad (3)$$

$$\hat{\beta} = \frac{1}{T}\sum_{t=1}^{T}(r_t - (\hat{\boldsymbol{\theta}}^T, \hat{\alpha})\boldsymbol{u}_{t-1})^2 = \frac{1}{T}\sum_{t=1}^{T}\left(r_t - \hat{\alpha} - \sum_{j=1}^{m}\hat{\theta}_j r_{t-t_j}\right)^2 \qquad (4)$$

where

$$\boldsymbol{u}_{t-1}^T = (\boldsymbol{z}_{t-1}^T, 1)\ \text{, and}\ \boldsymbol{z}_{t-1} = (r_{t-t_1}, \cdots, r_{t-t_m})^T$$

Also $\hat{\boldsymbol{\theta}}$ can be obtained from the well-known Yule-Walker equation which is described in section 4, and then $\hat{\alpha}$ can be written as

$$\hat{\alpha} = \frac{1}{T}\sum_{t=1}^{T}(r_t - \hat{\boldsymbol{\theta}}^T \boldsymbol{z}_{t-1}) = \bar{r}\left[1 - \sum_{j=1}^{m}\theta_j\right] \qquad (5)$$

This Equation shows that α is proportional to the mean radius vector length, $\bar{r}$ and is therefore a descriptor of the shape size. For large T, this LS estimates tend to ML estimates asymptotically.

The boundary representation and modeling schemes that we have explained so far, are insensitive to object translations, and to rotation of the object and variations in the starting sample over angles that are integral multiples of $2\pi/T$. It can also be shown that the CAR model Parameters, $\boldsymbol{\theta}$, are not dependent on the size of the object (refer to [3] for the proof), where as the parameters α and $\sqrt{\beta}$ are directly proportional to size. However the function $\alpha/\sqrt{\beta}$, which can be interpreted as a signal-to-noise ratio, is size independent. These invariant properties of the parameters $(\boldsymbol{\theta}, \alpha/\sqrt{\beta})$ make them attractive candidates as shape features for recognition purposes. This idea is realized in section 3.1.

3.Classification in noise-free environment

Having constructed the mathematical shape models, we are now able to develop a shape classifier based on the features that are extracted from the shape boundaries. The feature extraction procedure is described in section 3.1. Two fast classification algorithms which are suitable for near real-time applications are introduced and examined using a number of realistic machine parts and aircraft silhouettes, and a set of four letters of alphabet. All these three pattern sets are shown in Figs. 8, 9, and 10. At last the ability of the algorithms in recognizing deformed shapes is assessed in section 3.4.

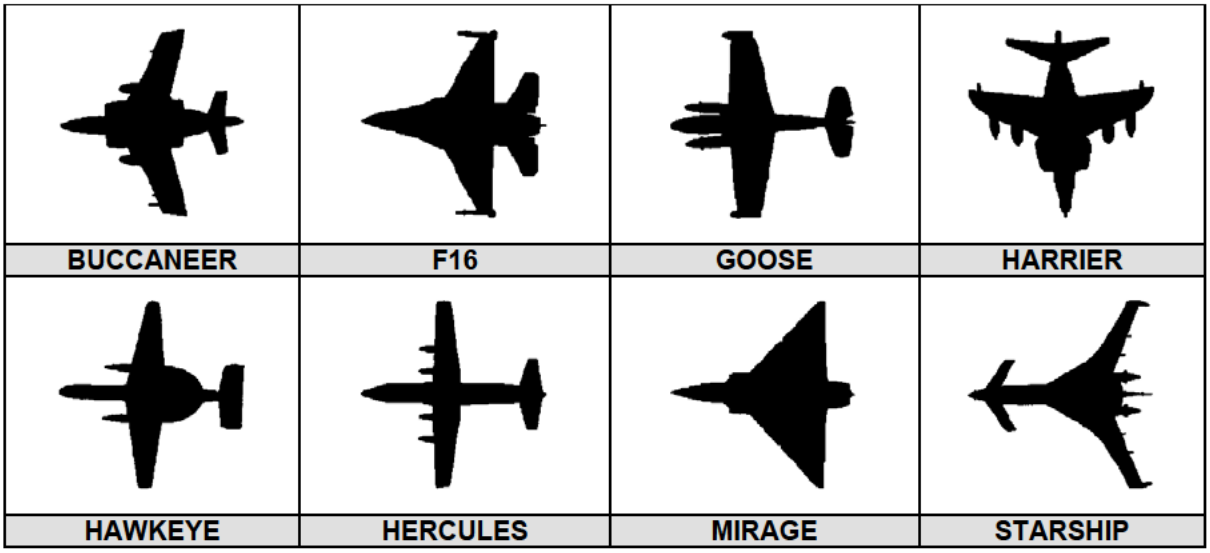


Fig. 8. Aircrafts

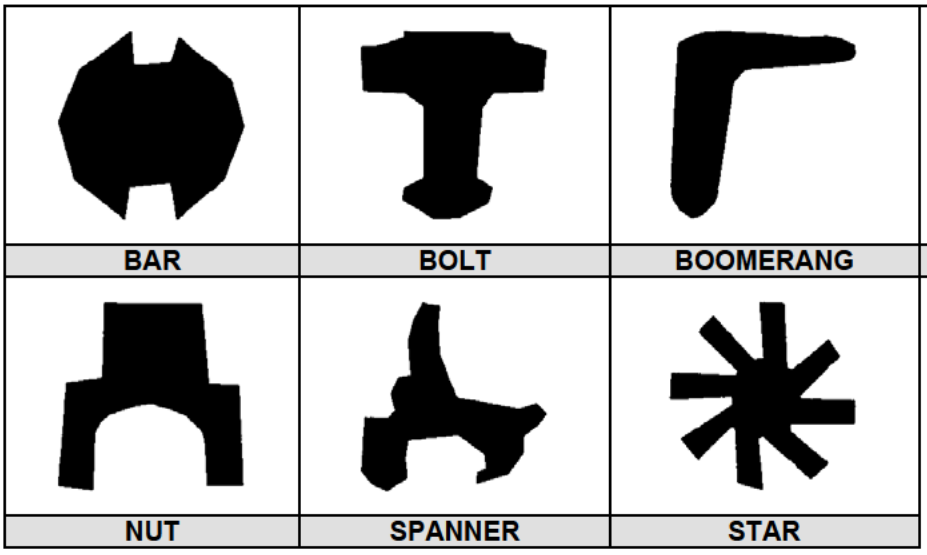


Fig. 9. Machine parts

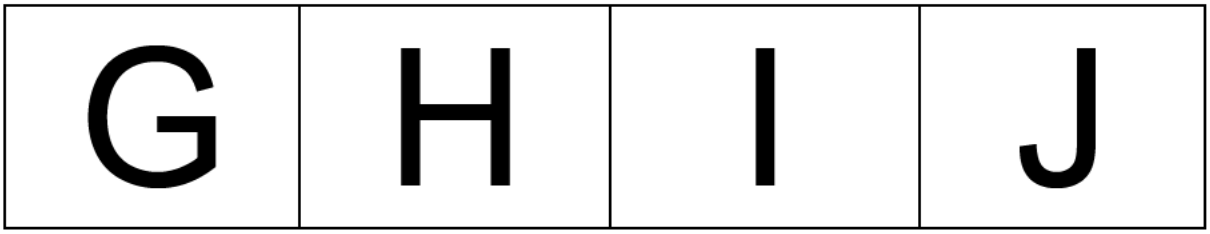


Fig. 10. Letters of alphabet (with font 'Arial')

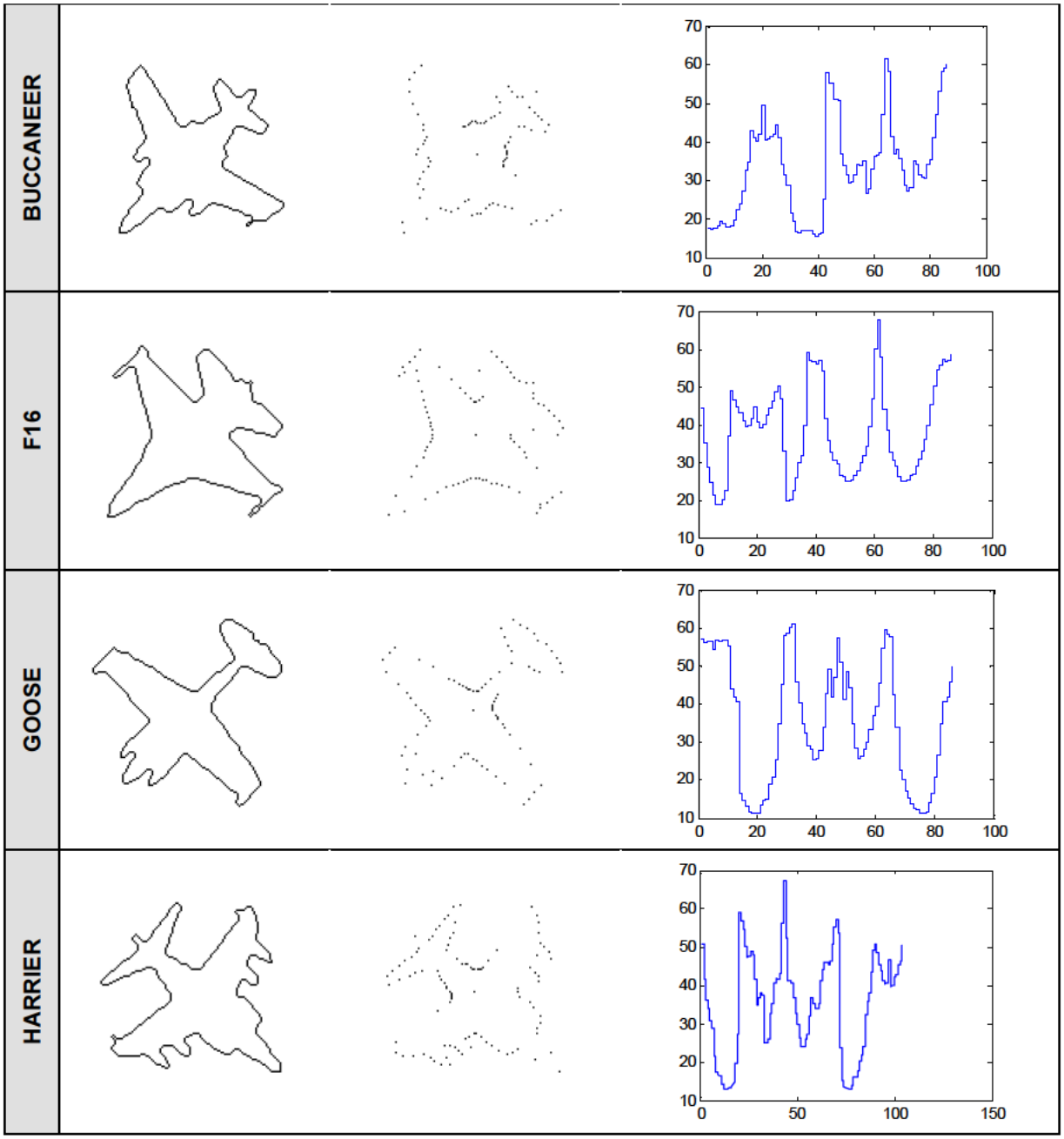
3.1. Feature extraction

In order to extract features from a given shape, we first represent its boundary by a one-dimensional time series. This procedure was discussed thoroughly in sections 2.1 and 2.2 and is summarized graphically in Fig. 11 for a scaled and 45° rotated version of aircraft templates shown in Fig. 8. Then a CAR model is fitted to the data and the parameters $(\theta, \alpha/\sqrt{\beta})$ are estimated. Then these parameters,

because of their special invariant properties are chosen as features defining the feature vector:

$$x = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_m \\ \alpha/\sqrt{\beta} \end{bmatrix}$$

We assume that rotations of the object over angles that are not integral multiples of $2\pi/N$ or variations in the starting sample point will produce similar time series if the number of radius vectors is large enough to accurately represent the boundary. Table 1 shows data for different sizes and rotations of 'Starship' chosen from aircraft templates in Fig. 8. The CAR model (of order 2) parameters were estimated from the lengths of $N = 64$ angularly equispaced radius vectors projected between the boundary and its centroid. In first three rows of table 1, the shape is scaled by different factors without any rotation. In rows 4 – 8, size is kept fix and the shape is rotated first through three different degrees (45˚, 90˚, and 315˚) that are integral multiples of $2\pi/N$ and then through two other degrees (240˚ and 120˚) that are not so. This data shows that the CAR model parameters preserve intraclass shape similarities among samples differing in size and spatial orientation, and are therefore suitable shape representatives for a classification task.



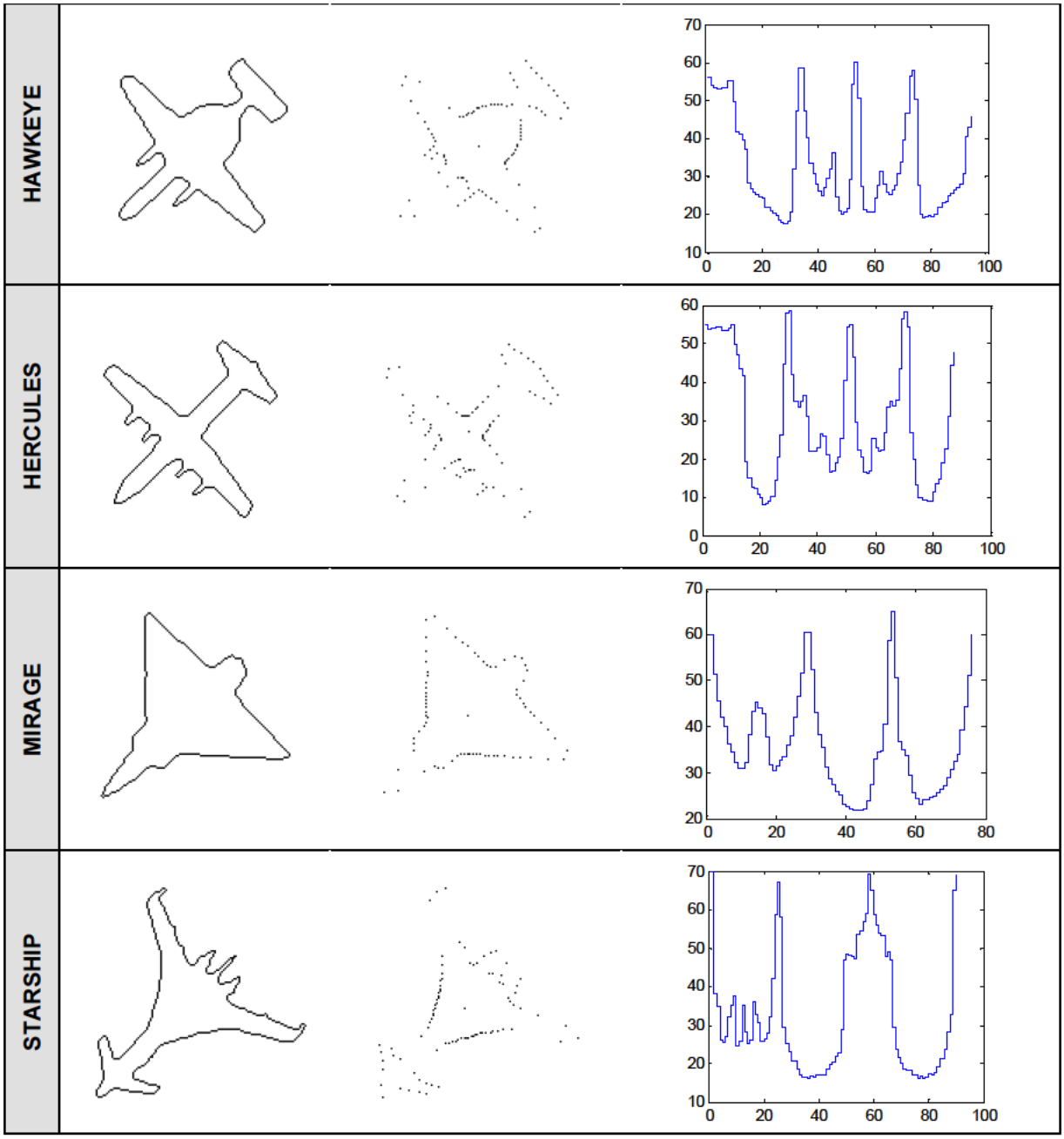


Fig. 11. Example of boundary extraction, boundary sampling and time series generation for scaled and 45 degrees rotated aircraft templates.

Table 1
Data for 'Starship' using $N = 64$ equiangular radius vectors and a CAR model of order m=2.

Rotation degree	Scale factor	T	θ_1	θ_2	α	β	$\alpha/\sqrt{\beta}$	Centroid x	Centroid y
0	0.7	72	1.0901	-0.2942	9.0121	130.84	0.7879	84	106
0	1.3	72	1.0633	-0.2592	16.2236	459.72	0.7567	156	196
0	1	74	1.1320	-0.3250	12.3873	244.57	0.7921	120	152
45	1	73	1.1399	-0.3248	11.9282	249.91	0.7545	153	193
90	1	74	1.0684	-0.2539	11.9474	258.91	0.7425	90	120
315	1	72	1.0847	-0.2879	12.9396	274.20	0.7813	193	193
240	1	76	1.1685	-0.3473	11.6590	219.2231	0.7874	193	152
120	1	76	1.1580	-0.3551	12.8931	248.00	0.8187	143	151

3.2. Classification schemes

In order to investigate shape classification by the use of CAR model parameters, we implement two simple and sufficiently fast classifiers: the Bayes optimal classifier with Gaussian parametric estimation of PDFs, and a linear classifier based on $L(L\text{-}1)/2$ hyper planes(L= number of classes). These classifiers are chosen because of their high recognition speed and their plausible accuracy. We do not intend to explain these classification algorithms here, as they can be found in most pattern recognition books like [4].

3.3. Tests and results

In this section all the algorithms that we have introduced so far are implemented with MATLAB programs. The sources of all these programs are given in the Appendix. For each shape we collect training and test data sets composed of samples of different sizes of the shape in various rotational positions. The size of each object image varies from 0.7 to 1.3 times the size of the original object. The shapes are then rotated over the range from 0 to 360 degrees. 76 angles for aircrafts and 64 angles for machine parts and letters are used to extract the data.

Table 2 shows the classification results for each pattern set. Classification performance is measured by the number of correctly classified pattern set samples divided by the total number of samples tested. In order to obtain these results, we modeled all of the shapes in each pattern set with a full CAR model of specific order and measured the performance of the two recognition algorithms introduced in section 3.2. We repeated the procedure for model orders 2 to 10 in order to obtain a measure of recognition performance versus model order. The relationship between model order and shape complexity suggests a possible connection between recognition performance and model order. There is also a relationship between model order and training set size. The Bayes optimal method can not be calculated (dashed lines in table 2) when the number of training set samples for a given class is less than or equal to m+1 because the sample covariance matrix of the class is singular. Therefore, the training set of each class must contain at least m+2 samples and m+1 of them must be linearly independent.

The results in table 2 indicate that the CAR model parameters are useful shape descriptors for recognition purposes, and the approximate boundary representation algorithm retains sufficient shape information for the estimation of these CAR model parameters. As we see in table 2, lots of our tests show perfect (100%) classification results. The results are dependent on the particular model order used and on the classification scheme employed. We can also see that the classification accuracy improves as the number of training samples increases. It is quite surprising that we obtain

Table 2
Average correct classification percentages for pattern sets shown in Figs. 8, 9, and 10.

Model order	Method	Aircrafts			Machine Parts			Letters (Arial)		
		N_{tr}=10	N_{tr}=20	N_{tr}=30	N_{tr}=10	N_{tr}=20	N_{tr}=30	N_{tr}=10	N_{tr}=20	N_{tr}=30
2	Bayes	100	100	100	98.86	99.43	99.43	99.5	98.5	99.5
	Linear	*99.25*	*100*	*100*	*98.29*	*96.86*	*99.43*	*100*	*100*	*100*
3	Bayes	100	100	100	95.71	99.43	99.71	99	99.5	99.5
	Linear	*98.25*	*100*	*100*	*97.71*	*99.71*	*99.71*	*99*	*100*	*100*
4	Bayes	100	100	100	92.29	99.71	100	100	100	99.5
	Linear	*99.75*	*99.5*	*100*	*97.71*	*99.71*	*100*	*97.5*	*100*	*99*
5	Bayes	100	100	100	90	100	100	99	100	100
	Linear	*99*	*99.75*	*100*	*96*	*98.57*	*99.71*	*98*	*99*	*100*
6	Bayes	100	100	100	84	100	100	98.5	100	100
	Linear	*98*	*99.25*	*99.75*	*89.43*	*98.29*	*99.43*	*95.5*	*99.5*	*100*
7	Bayes	100	100	100	74.57	98	99.71	94.5	99.5	100
	Linear	*98.75*	*100*	*99.75*	*89.71*	*97.43*	*99.14*	*95*	*98.5*	*100*
8	Bayes	78.5	100	100	52.29	98.29	100	94	100	100
	Linear	*97.25*	*99.75*	*99.5*	*86.86*	*98*	*99.14*	*95*	*98.5*	*99.5*
9	Bayes	---	100	100	---	96.57	99.71	---	100	100
	Linear	*92*	*99.5*	*99.75*	*75.71*	*94.57*	*99.43*	*88.5*	*98.5*	*99.5*
10	Bayes	---	100	100	---	96.29	99.71	---	99.5	99.5
	Linear	*91.5*	*99.25*	*99.5*	*79.43*	*96.29*	*96.29*	*85.5*	*97.5*	*98.5*

- The average probability of error in Bayes classifier is zero in all tests.
- No sample is assigned to unknown class in any tests with linear classifier.

perfect classification results for aircrafts with only 10 training samples for model orders 2 – 7. At model order 5 and for $N_{tr} \geq 20$, the Bayes optimal classifier shows 100 percent correct classification results for all pattern sets.

Generally we can say that in our problem, the Bayes classifier performance is better than the linear classifier performance. However, for small sizes of training set (e.g. $N_{tr} = 10$), the linear classifier is less sensitive to changes in model order than the Bayes classifier and also it does not have the particular singularity problem that we encounter in Bayes classifier when N_{tr} is small. Thus, the linear classifier can be used when training set size prohibits the use of the Bayes classifier.

Another point concluded from the results is that the shapes can be represented by CAR models of order lower than the optimum for these shapes, and can still be successfully classified by the Bayes optimal classifier. Thus, the model order of each individual shape sample does not have to be determined for accurate recognition.

3.4. Examining the ability of classification algorithms in recognizing deformed shapes

In a classification problem, there might be some special situations in which the shape of the objects changes in different ways. One example of such special problems, which is actually one of the most difficult problems in recognition, is the classification of partial views of shapes based only on knowledge of the whole shapes. Our described algorithms were tested in this situation in [2] by artificially covering portions of the shapes and recognition accuracies of 72.5% to 85% for model orders 1– 9 were obtained (the Rotated Coordinate System classifier was used in [2] instead of our Bayes classifier). These results show that the CAR model parameters can be useful features for partially occluded shapes, but further study is needed to determine the practicality of the application.

Here we test another possible kind of shape deformation through the set of letters of alphabet. This is the situation in which the classifier is trained using the letters written with a specific font while the test letters are in some other fonts. As shown in Fig.12, in our experiment we train the classifier using the font 'Arial' and then test it using letters with the three fonts: 'Times New Roman', 'Book Antiqua', and 'Tahoma'. Table 3 summarizes the results obtained by the use of the Bayes optimal classifier. As we see, these results are actually much worse than what we expected! The maximum correct classification for 'Book Antiqua' is 52%, for 'Times New Roman' is 57%, and for 'Tahoma' is 66% which are obtained from model orders 5, 5, and 3 respectively. Also, as the last row of table 3

ARIAL	TIMES NEW ROMAN	BOOK ANTIQUA	TAHOMA
G	G	G	G
H	H	H	H
I	I	I	I
J	J	J	J

Fig. 12. Probable deformation in letters set.

indicates, the correct classification rate decreases as the number of training data is increased. This shows that the ability of the classifier in recognizing the deformed shapes reduces as the classifier learns more about the original shape through more training samples.

The results for the linear classifier have not been included in table 3 as they vary significantly each time we run the program. As we know, In the linear classifier we consider random initial values for the normal vector of each hyper plane. Since these values vary in each run of the program, the results may not be the same for all runs. In our previous tests these changes were small and negligible but here, in the case of deformed shapes, we see much variation in the results and correct classification percentages from 30% up to 80% may be obtained for a single template! Thus we did not include these results in table 3.

Table 3
Average correct classification percentages for deformed pattern sets shown in Fig. 12.

Model order	Method	Book Antiqua			Times New Roman			Tahoma		
		N_{tr}= 10	N_{tr}= 20	N_{tr}= 30	N_{tr}= 10	N_{tr}= 20	N_{tr}= 30	N_{tr}= 10	N_{tr}= 20	N_{tr}= 30
2	**Bayes**	38	24	25	49	30	29	49	45	46
3	**Bayes**	29	28	27	33	47	43	**66**	65	65
4	**Bayes**	27	27	25	33	29	32	54	52	53
5	**Bayes**	**52**	46	38	**57**	45	26	50	52	53
6	**Bayes**	45	38	32	57	39	25	47	49	49
7	**Bayes**	43	38	37	51	26	23	50	49	49
8	**Bayes**	35	46	39	48	29	26	50	50	48
Maximum		**52**	46	39	**57**	47	43	**66**	65	65

4. Classification in noisy outlier contaminated environment [1]

Through the remainder of this project we consider the problem of classifying objects in environments generating significant imaging noise. The latter increasing the likelihood of contamination by large aberrant observations (outliers). These are typically generated by failures in boundary extraction or long tailed imaging noise. As same as previous sections our focus is on near real-time classification algorithms. Typical examples are the automatic inspection of industrial components, where the aim is to identify and characterize faults, autonomous vehicle navigation, where unexpected objects enter the domain, and the analysis of speech spectrograms. For extracting information from outlier contaminated data we use techniques which are insensitive to outliers. An alternative approach based on the direct use of the spectral function estimated using sub-set auto-regression is introduced and benchmarked against the feature based approach used in noise-free environment. It is shown that we achieve a significant performance improvement by the use of sub-set AR models instead of full models.

4.1. The model

Let $X=(X_t, t=1,\ldots,T)$ be an estimated time series of a given object. We construct a noisy outlier contaminated version $Y=(Y_t, t=1,\ldots,T)$ of X using

$$Y = X + N + k\sigma_X A \tag{6}$$

where $N^T=(N_t, t=1,\ldots,T)^T$ is a vector of independent normally distributed random variables with zero mean and variance σ_N^2. A^T is a vector with T- int (sT) zero elements at random locations (int(sT) means the integer part of (sT)), while the remainder take the value +1 or -1 with equal probability. In this way we introduce int(sT) outliers of magnitude $+k\sigma_X$ or $-k\sigma_X$. This generates outliers with similar relative size for all templates, as σ_X^2 is the variance of X.

Following Kashyap and Chellappa [3], we assume that Y is generated by a circular sub-set auto regression $CSAR(J,\mu,\phi_J,\sigma_Z)$ of the form

$$Y_t-\mu=\sum_{j\in J}\phi_J(j)(Y_{[t-j]}-\mu)+Z_t,\quad t=1,\cdots,T \tag{7}$$

where $(j; j\in J)$ are 'time' lags in the model, $\phi_J=(\phi_J(j), j\in J)^T$ the corresponding parameters, μ a location parameter and $(Z_t, t=1,\ldots,T)$ the innovations. We assume that the latter are a sequence of independent, normally distributed random variables with zero mean and variance σ_Z^2.

This is the same model we used in section 2.3 where μ is equivalent to α and σ_Z^2 is equivalent to β. In noise-free environment we considered a full model of order $m, (J = (1,2,\ldots,m)^T)$. Here, in the case of sub-set model, we employ a robust lag selection procedure to find a near optimum lag structure J for each template. This is presented in section 4.3. Particular robust parameter estimations are also introduced in section 4.2 to mitigate the effect of outliers.

4.2. Robust parameter estimation

Consider a circular sub-set auto regression with lags J and parameters $\phi_J = (\phi_{J(j)}, j \in J)^T$. We estimate the latter from a single data record $Y = (Y_t, t = 1,\ldots,T)$ using the Yule-Walker equations. These are $\hat{\boldsymbol{R}}_J \hat{\phi}_J = \hat{r}_J$, where $\hat{\boldsymbol{R}}_J = (\hat{r}(|i-j|), i, j \in J)$ and $\hat{r}_J = (\hat{r}(i); i \in J)$. When non-robust procedures are required, $(\hat{r}(u), u \geq 0)$ is the sample covariance function

$$\hat{r}(u) = \frac{1}{T}\sum_{t=1}^{T}(Y_t - \hat{\mu})(Y_{[t+u]} - \hat{\mu}) \tag{8}$$

In this case the estimated parameters $\hat{\phi}_J$ are exactly equal to what is obtained by the formulas presented in section 2.3. Here, to mitigate the effect of outliers, we use the robust estimate of the auto covariance function given by

$$\hat{r}(u) = \hat{\sigma}_X^2 \frac{\hat{c}(u)}{\hat{c}(0)}; \quad u \geq 0 \tag{9}$$

where

$$\hat{c}(u) = \frac{1}{T}\sum_{t=1}^{T}\psi_{H,f}\left(\frac{Y_t - \hat{\mu}}{\hat{\sigma}_X}\right)\psi_{H,f}\left(\frac{Y_{[t+u]} - \hat{\mu}}{\hat{\sigma}_X}\right). \tag{10}$$

For non-robust procedures, $\hat{\mu}$ is estimated by the sample mean value $\overline{Y}$, and the sample standard deviation is used for $\hat{\sigma}_X$. The robust alternatives for these parameters are

$$\hat{\mu} = \text{median}(Y_t, t = 1,\cdots,T), \tag{11}$$

$$\hat{\sigma}_X = 1.483\,\text{median}\left(|Y_t - \hat{\mu}|, t = 1,\cdots,T\right). \tag{12}$$

$\psi_{H,f}(x)$ is Huber's function

$$\psi_{H,f}(x) = \begin{cases} x & |x| \leq f \\ f\,\text{sign}(x) & |x| > f \end{cases} \tag{13}$$

where f is a parameter chosen appropriately for each template.

We take the usual regression estimate

$$\hat{\sigma}_Z^2 = \left(\frac{T - \text{card}(j)}{T - 2\text{card}(j) - 1}\right)\left(\hat{r}(0) - \sum_{j \in J}\hat{\phi}_J(j)\hat{r}(j)\right), \tag{14}$$

where card(j) is the number of lags in the model. This formula differs from (4) by the factor

$$\left(\frac{T - \text{card}(j)}{T - 2\text{card}(j) - 1}\right),$$

which is nearly equal to 1 when T is large enough.

4.3. Robust lag selection

In this section we show how the lags of a sub-set auto regression can be determined from N multiple data records $(Y_k, k = 1,\ldots,N)$. The kth record $Y_k = (Y_{k,1},\ldots,Y_{k,T_k})$ has T_k elements. In our experiment there are $N = N_i$ data records associated with the ith class.

The robust Yule-Walker equations are used to estimate the parameters of candidate models in each data record. These estimates averaged over N data records make the class parameters $\tilde{\sigma}_Z^2$ and $\tilde{\phi}_J$

$$\tilde{\sigma}_Z^2 = \frac{1}{N}\sum_{k=1}^{N}\left(\hat{\sigma}_Z^{(k)}\right)^2 \tag{15}$$

$$\tilde{\phi}_J = \frac{1}{N}\sum_{k=1}^{N}\hat{\phi}_J^{(k)}. \tag{16}$$

Let the lags in the model at the vth iteration be denoted by J_v. We calculate the parameter estimates $\tilde{\phi}_{J_v}$ using the equation (16) and section 4.2. Then we determine the lags $J_{v+1} = J_v \backslash j$ (lags in J_v with j removed) which minimize the loss function

$$\delta^{\beta}(J_v, j) = \beta \log \tilde{\phi}_{J_v}^2(j) + (1-\beta)\log D(J_v, j) \tag{17}$$

where

$$D(J_v, j) = \frac{1}{N}\sum_{k=1}^{N}\hat{r}_k(0)\left(\hat{\phi}_{k,J_v}(j) - \tfrac{1}{N}\sum_{l=1}^{N}\hat{\phi}_{l,J_v}(j)\right)^2 \tag{18}$$

The factor β in equation (17) is used to balance the effect of the two terms in the loss function. The formula used in [1] for $D(J_v, j)$ is

$$D(J_v, j) = \sum_{k=1}^{N}\hat{r}_k(0)\hat{\boldsymbol{R}}_{k,J_v}^{-1}$$

where $(\hat{R}_{k,J_v} = \hat{r}_k(u,v), (u,v) \in J_v)$ and $(\hat{r}(u), u \geq 0)$ an

estimate of the auto covariance function using $Y=Y_k$, see section 4.2. This formula seems to be incorrect or some printing mistake might have occurred in the original paper. In this formula $\hat{r}_k(0)=\hat{\sigma}_X^{(k)}$ is a single value and $\hat{\boldsymbol{R}}_{k,J_v}$ is a matrix, so $D(J_v, j)$ becomes a matrix while the loss function involves it being a single value. Thus, considering the main idea used in the definition of the loss function, we suggest the alternative formula (18). The first term in the loss function indicates that it is desirable to omit the lag j which its corresponding coefficient $\tilde{\phi}_{J_v}(j)$ has the smallest absolute value among all lags coefficients and has therefore the least effect in constructing the model. In the second term through the use of the formula suggested for $D(J_v, j)$ we panelize lags associated with parameter estimates with large variance. If we ignore the factor $\hat{r}_k(0)$ which is equal for all candidate lag structures, $D(J_v, j)$ is equal to the variance of $\hat{\phi}_{J_v}(j)$. In order to understand the role of the term $D(J_v, j)$, consider the lags j' and j'' whose corresponding parameters $\tilde{\phi}^2_{J_v}(j')$ and $\tilde{\phi}^2_{J_v}(j'')$ are smaller than all the other parameters, and $\tilde{\phi}^2_{J_v}(j') \le \tilde{\phi}^2_{J_v}(j'')$. Now suppose we compute D for these lags and see that $D(J_v, j')$ is much larger than $D(J_v, j'')$. This shows that although $\tilde{\phi}^2_{J_v}(j') \le \tilde{\phi}^2_{J_v}(j'')$, the certainty of the estimation for $\tilde{\phi}_{J_v}(j')$ is lower than for $\tilde{\phi}_{J_v}(j'')$ as it has larger variance. So, it is more plausible to omit j'' instead of j'. The second term in the loss function through the use of the tuning factor β gives us the flexibility to change the decision in such cases.

The procedure described above is repeated for $v=1,2,\ldots,m-1$ where m is the number of lags in the initial model (a full auto regression). Finally, the lags of the sub-set auto regression generating the data $(Y_k, k=1,2,\ldots,N)$ is identified by

$$\hat{J}=\arg\min_{J} RFPE(J), \quad J=J_0, J_1, \cdots, J_{m-1} \tag{19}$$

where *RFPE(J)* is a robust analogue of the *final prediction error* criterion for multiple data records. Here

$$RFPE(J)=\tilde{\sigma}_Z^2(J)\left(1+\gamma\frac{|J|+1}{\sum_{k=1}^{N}T_k}\right) \tag{20}$$

where $|J|$ is the number of lags in J, $\tilde{\sigma}_Z^2(J)$ is the estimate of the innovation variance given by (15), and γ is chosen equal to 1.

4.4. Spectral function estimation and the classification scheme

In this section we describe the recognition scheme based on the spectral functions estimated from CSAR model parameters. We assume that Y is standardized by an estimate of the scale of X, $\hat{\sigma}_X$ which as stated before, is the sample standard deviation in non-robust procedures or is estimated by equation (12) for robust algorithms.

First we estimate the spectral functions associated with the C object classes using the training data. The ith class contains N_i independent records. We obtain the CSAR model parameters for the kth record in the ith class for all values of k. Then we calculate the smooth interpolant of the spectral function associated with the ith class by

$$\tilde{\mathrm{f}}_i(\lambda)=\tilde{\sigma}_Z^2\,\mathrm{g}(\lambda,\tilde{\phi}_J), \quad 0\le\lambda\le 2\pi \tag{21}$$

where

$$\mathrm{g}(\lambda,\tilde{\phi}_J)=\frac{1}{2\pi\left|1-\sum_{l\in J}\tilde{\phi}_J(l)e^{-\mathrm{i}\lambda l}\right|^2} \tag{22}$$

with estimates of σ_Z^2 and ϕ_J given by (15) and (16) respectively. It is easy to see that $\tilde{\mathrm{f}}_{\mathrm{i}}(\lambda)$ is invariant to shifts, rotation, and scale changes when Y is standardized.

Now, we use these spectral functions to classify objects. A distance based classifier is suggested for this case in [1] as it appears to give better performance than feature based techniques. The distance is defined here as

$$\bar{d}_i^{\,w}=d^w(\tilde{\mathrm{f}}_i,\hat{\mathrm{f}})+d^w(\hat{\mathrm{f}}_{\mathrm{i}},\tilde{\mathrm{f}}), \tag{23}$$

where

$$d^w(\mathrm{f},\mathrm{g})=\left(\frac{1}{\sum_{t=0}^{T-1}w(\lambda_t)}\right)\sum_{t=0}^{T-1}w(\lambda_t)\,\mathrm{h}\left(\frac{\mathrm{f}(\lambda_t)}{g(\lambda_t)}\right), \tag{24}$$

$\lambda_t=2\pi t/T$.

$\hat{\mathrm{f}}(\lambda)$ is an estimate of the spectral function associated with Y that is the observation being classified. This is derived from CSAR model fitted to Y and spectral function estimated for it considering $N=1$ in (15) and (16) which leads to

$\tilde{\sigma}_Z^2=\hat{\sigma}_Z^2$ and $\tilde{\phi}_J=\hat{\phi}_J$.

In our experiment we take $h(x) = (x\text{-}1)^2$ and $w(\lambda) = 1$ for values of λ that construct the most frequency content of the spectral functions and zero elsewhere. We will show that most differences between spectral functions are contained in the interval $0 \le \lambda \le 2\pi l/100$, where $l = 20$. By taking $h(x) = x$ and $w(\lambda) = 1$, we obtain the familiar quadratic descriminant.

Finally, we allocate Y to class v if

$$\bar{d}_i^w > \bar{d}_v^w, \quad i \neq v \ . \tag{25}$$

4.5. Tests and results

In this section, we examine the performance of the suggested algorithms using the same pattern sets of section 3. We first generate a primary version of training and test data through randomly scaling (0.7~1.3) and rotating (0°~360°) each template about the center of gravity of its vertices and sampling its boundary using 76 angles. We consider 100 records for each of test and training data sets. Contamination is then added to this data using Eq. (6). We take $\sigma_N^2 = 4^2$, $s = 0.02$, and $k = 0, 12$. This ensures that there are (generally) one or two outliers (s = 0.02) in each profile when $k \neq 0$. To illustrate the nature of the contamination used in our experiment we plot the time series associated with the 'Harrier' before noise addition and after that for k = 0 and k = 12 in Fig. 13. As stated in this figure, parameters $\hat{\mu}$ and $\hat{\sigma}_X$ estimated by Eqs. (11) and (12) are almost insensitive to outliers. It must be noted here that for data with large variance σ_X, outliers of magnitude $-k\sigma_X$ can result in negative values for the time series. As this is not realistic (radius vector lengths can not have negative values), we cut off these negative peaks as shown in Fig. 13.

To be able to use the robust parameter estimation algorithm described in section 4.2, we must first choose an appropriate value for Huber parameter f. In [1], the same value $f = 2$ has been taken for all aircraft and machine part templates. Here we try to find better values for f associated with each template. Consider a given data record Y. In the absence of outliers ($s = 0$ or $k = 0$), the parameter estimates obtained from robust algorithms are desired to be the same as those obtained from non-robust algorithms. It can be seen in Eqs. (9) and (10) that if we choose f greater than or equal to $(Y_t - \hat{\mu})/\hat{\sigma}_X$ for all values of t $(t = 1,2,\ldots,T)$, then $\hat{r}$ will become equal to the sample covariance function and the two algorithms will be equivalent. However, large values of f have little mitigation effect in the presence of outliers. Thus, for this given record Y, we choose f exactly equal to the maximum value of $(Y_t - \hat{\mu})/\hat{\sigma}_X$ over all values of t.

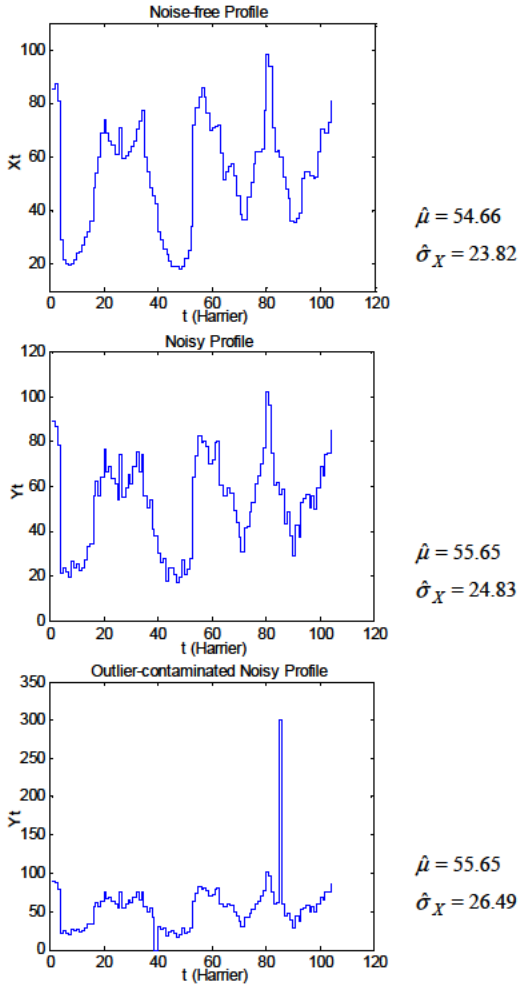


Fig. 13. Time series for Harrier before noise addition and after that for ($s = 0$) and ($s = 0.02$, $k = 12$).

In our experiment, since we have $N_i = 100$ data records for each template in the training set, we choose the value of f for each template i as

$$f^{(i)} = \max\left\{ \max\left\{ \frac{Y_{tk}^{(i)} - \hat{\eta}_k^{(i)}}{\hat{\sigma}_{X_k}^{(i)}}, t = 1,2,\cdots,T \right\}, k = 1,2,\cdots,N_i \right\} \tag{26}$$

These values, obtained for all templates of each pattern set, are given in table 4.

Table 4.
Huber parameters.

Aircrafts	f	Machine Parts	f	Letters(Arial)	f
Buccaneer	2.64	Bar	10	G	4.35
F16	2.58	Bolt	1.57	H	2.38
Goose	1.54	Boomerang	2.15	I	8.23
Harrier	2.35	Cylinder	1	J	3.4
Hawkeye	3.72	Nut	4		
Hercules	2.4	Spanner	1.62		
Mirage	3.87	Star	1.87		
Starship	4.92				

Another step is still left to make the parameter estimation algorithms completely ready to use. This step is the determination of lag structures associated with templates of each class. This lag selection is carried out independently for noisy data ($k = 0$) and contaminated data ($k = 12$, $s = 0.02$) using the robust algorithm described in section 4.3 and taking $m = 30$ for aircraft silhouettes, and $m = 20$ for machine parts and letters. In both cases we use the loss function δ^{β} with $\beta = 0.2$, although different values of β have little effect on the performance of our algorithm, as the number of samples in the training set is large. The lags obtained in our experiment are given in tables 5 and 6.

Table 5
Lags identified by the robust procedure described in section 4.3 for silhouettes of Figs. 8, 9, and 10 using outlier free data

	Templates \ Lags	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Aircrafts	BUCCANEER	●	●									●									●	●	●	●	●	●	●	●			
	F16	●	●	●	●	●	●																	●	●	●		●	●		
	GOOSE	●	●	●	●	●												●	●	●	●										
	HARRIER	●	●	●	●				●																			●	●	●	●
	HAWKEYE	●	●															●	●	●	●			●	●						
	HERCULES	●	●											●	●	●	●		●		●	●	●	●	●						
	MIRAGE	●	●																					●	●	●					
	STARSHIP	●	●									●	●		●										●	●	●				
Machine Parts	BAR	●					●		●																						
	BOLT	●	●		●													●	●												
	BOOMERANG	●	●	●																											
	CYLINDER									●				●					●		●										
	NUT	●	●	●										●	●	●															
	SPANNER	●	●	●														●		●	●										
	STAR	●								●	●									●	●										
Letters	G	●	●	●	●						●																				
	H	●	●									●	●																		
	I	●	●	●	●																										
	J	●	●	●	●										●																

Table 6
Lags identified by the robust procedure described in section 4.3 for silhouettes of Figs. 8, 9,and 10 using outlier contaminated data with k=12

	Templates \ Lags	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Aircrafts	BUCCANEER	●						●				●	●	●							●	●	●	●	●						
	F16	●	●	●	●	●	●	●	●	●	●														●						
	GOOSE	●	●	●	●	●	●											●													
	HARRIER	●	●								●																	●	●	●	
	HAWKEYE	●	●																		●		●	●	●						
	HERCULES	●	●	●										●		●	●		●												
	MIRAGE	●	●				●																	●	●						
	STARSHIP	●	●						●	●											●			●	●						
Machine Parts	BAR	●	●																												
	BOLT	●	●		●	●												●													
	BOOMERANG	●	●	●																											
	CYLINDER													●							●										
	NUT	●	●	●						●																					
	SPANNER	●	●									●					●	●		●											
	STAR	●								●	●								●	●	●										
Letters	G	●	●	●	●	●							●																		
	H	●	●	●									●	●																	
	I	●	●	●	●																										
	J	●	●	●										●																	

4.5.1. Evaluating the robust parameter estimation algorithms through spectral functions

To evaluate the performance of the robust parameter estimation algorithm, we estimate and sketch the corresponding spectral functions for aircraft templates, see Figs. 14 and 15. The spectral function determined by robust techniques is denoted by $\tilde{f}_R$, see Eq. (21), with the corresponding non-robust estimate by $\tilde{f}_{YW}$. It is readily apparent that the aircraft silhouettes give a prominent low frequency peak in the spectral function. By comparing the solid and dashed lines in Fig. 14, we see that the non-robust estimates $\tilde{f}_{YW}$ are greatly distorted by the presence of outliers, with the spectral function becoming more like white noise. When robust procedures are used a different pattern emerges. Here $\tilde{f}_R$ is relatively unaffected by the presence of outliers, with similar shapes in outlier-free and contaminated data. By comparing Figs. 4 and 5 we see that robust and non-robust estimates are similar in outlier free data.

4.5.2. A classification experiment

In this section, we first generate and store the spectral function parameters $\tilde{\sigma}_Z$ and $(\tilde{\phi}_J(j), j \in J)$ associated with each class using the training time series. For test data, we estimate and store parameters $\hat{\sigma}_Z$ and $(\hat{\phi}_J(j), j \in J)$ for all test records of each class. These training and test data are the final data used for the classification task, and are generated independently for $k = 0$ and $k = 12$ with $s = 0.02$.

In order to benchmark this new approach (based on spectral functions) against Dubois and Glanz (DG) approach in [2] (feature based approach), we generate DG training and test data too, using the feature vector $(\phi_1, \phi_2, \ldots, \phi_M, \hat{\mu}/\hat{\sigma}_Z)^T$ where we take $M = 5$ which is the full model order.

To clarify the effect of robust algorithms, all the abovementioned training and test data are generated using both robust and non-robust procedures. We also examine the effect of excluding frequencies in the spectral function based classifier by taking $w(\lambda) = 1$ for $0 \le \lambda < 2\pi l/100$, with $l = 20$ and $l = 50$. In [1], $l = 100$ has been taken instead of $l = 50$. Because the portion of spectral functions located from $\lambda = 2\pi 50/100$ to $\lambda = 2\pi 100/100$ is the mirror image of the portion $\lambda = 0$ to $\lambda = 2\pi 50/100$, we take $l = 50$ instead of $l = 100$ to reduce the computations.

The percentages of correctly classified templates using 100 unclassified test templates for various levels of contamination (k=0 and k=12) are summarized in tables 7 – 9 for both robust and non-robust cases. Confusion matrices are also given in tables 10 – 27.

It must be noted that, on the contrary with [1], we do not use the lag selection procedure in the test set, as it reduces the speed of recognition task significantly so that it can not be comparable with DG approach. The same lag structures obtained for training records are used for test records too.

For outlier contaminated data ($s = 0.02$, $k = 12$) we see that non-robust procedures (based on sample covariance function) suffer large reductions in performance. This is in marked contrast to the suggested robust approach which is almost insensitive to outliers. For aircrafts, it is readily apparent that robust spectral method gives significantly better performance than DG approach with robust parameter estimation. For machine parts, again the spectral approach has better performance although the difference between two methods is not very big. For letters, on the contrary with aircrafts and machine parts, the performance of DG approach is better. This shows that for a given problem, we can not certainly say which of the two methods will give better results and it is better to test both methods, however , for problems in which the shapes are rather complex (such as aircrafts) the use of spectral classifier is preferable.

For outlier free data ($k = 0$) the relative performance of the techniques used in this study varies slightly according to the templates used, see tables 7, 8, and 9. In broad terms, non-robust techniques have a little better performance, with DG approach giving the best (or equal best) performance.

The use of weights with $l = 20$, has little overall effect on classification performance in outlier free data, although significantly improved performance in outlier contaminated data, see 'Nut' in table 8.

To gain further insight into the relative performance of the techniques under consideration we investigate selected confusion matrices. From tables 10 – 27, in the case of outlier contaminated data, we see that the confusion matrices associated with the non-robust spectral approach are greatly affected by the presence of outliers, although retaining some structure in common with their robust analogue. For example 'Spanner' in table 19 is most likely alternative to 'Bar'.

The confusion matrices of the spectral approach have similar structures in outlier free data for robust and non-robust procedures. Also increasing l to 50 has little effect on the corresponding confusion matrices.

In the case of DG algorithm, a similar pattern emerges, with the presence of outliers having a large effect on the structure of the confusion matrices of non-robust procedures. By comparing tables 14 and 15 we see that the structure of the confusion matrices associated with the spectral approach differs substantially from DG approach.

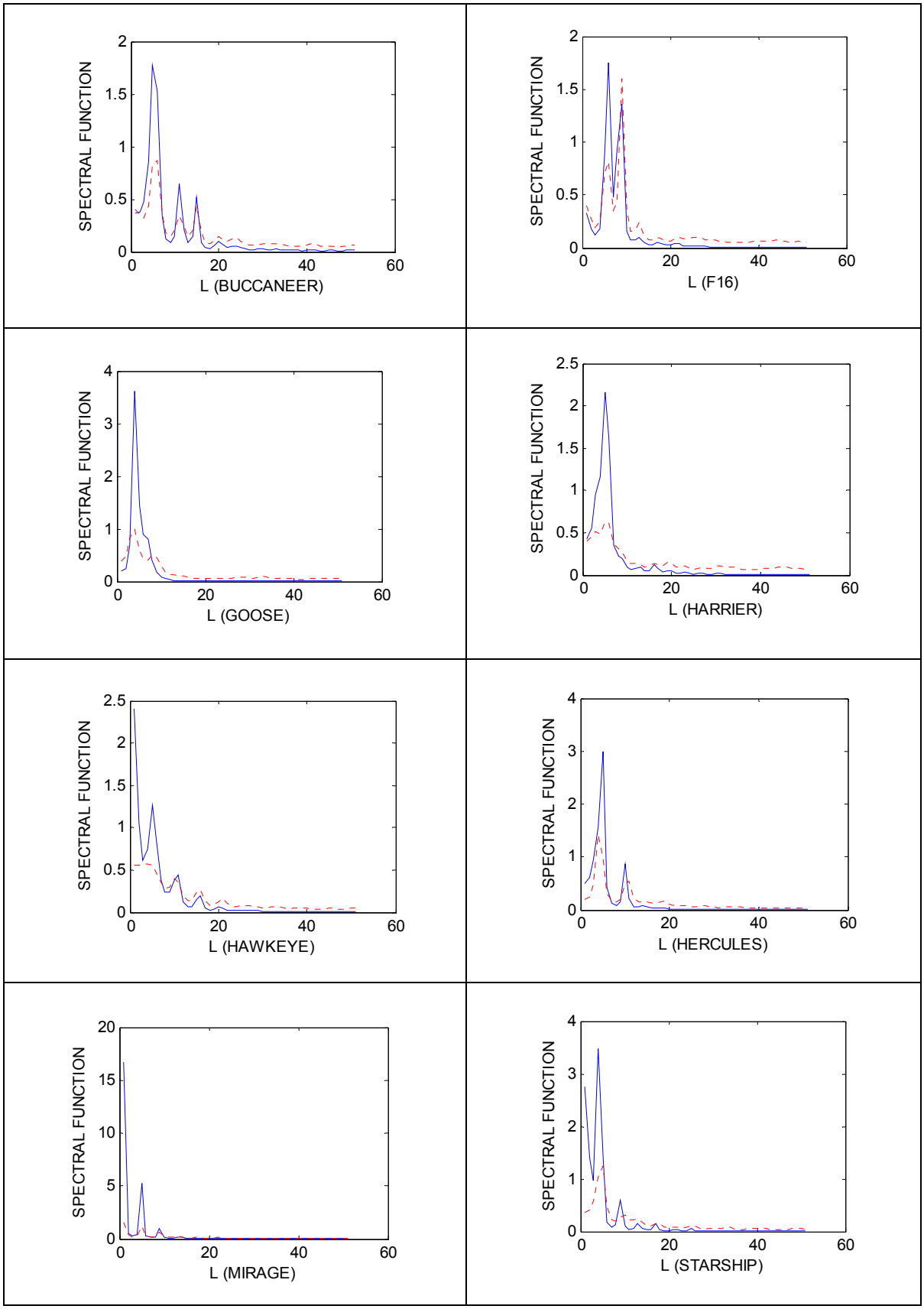


Fig. 14. Estimates of the spectral function $\tilde{f}_{YW}$ for the templates in Fig. 8. The solid line gives the non-robust estimate in outlier free data, with the dashed line its value in outlier contaminated data ($s = 0.02$, $k = 12$). Here L refers to the frequency $2\pi L/100$.

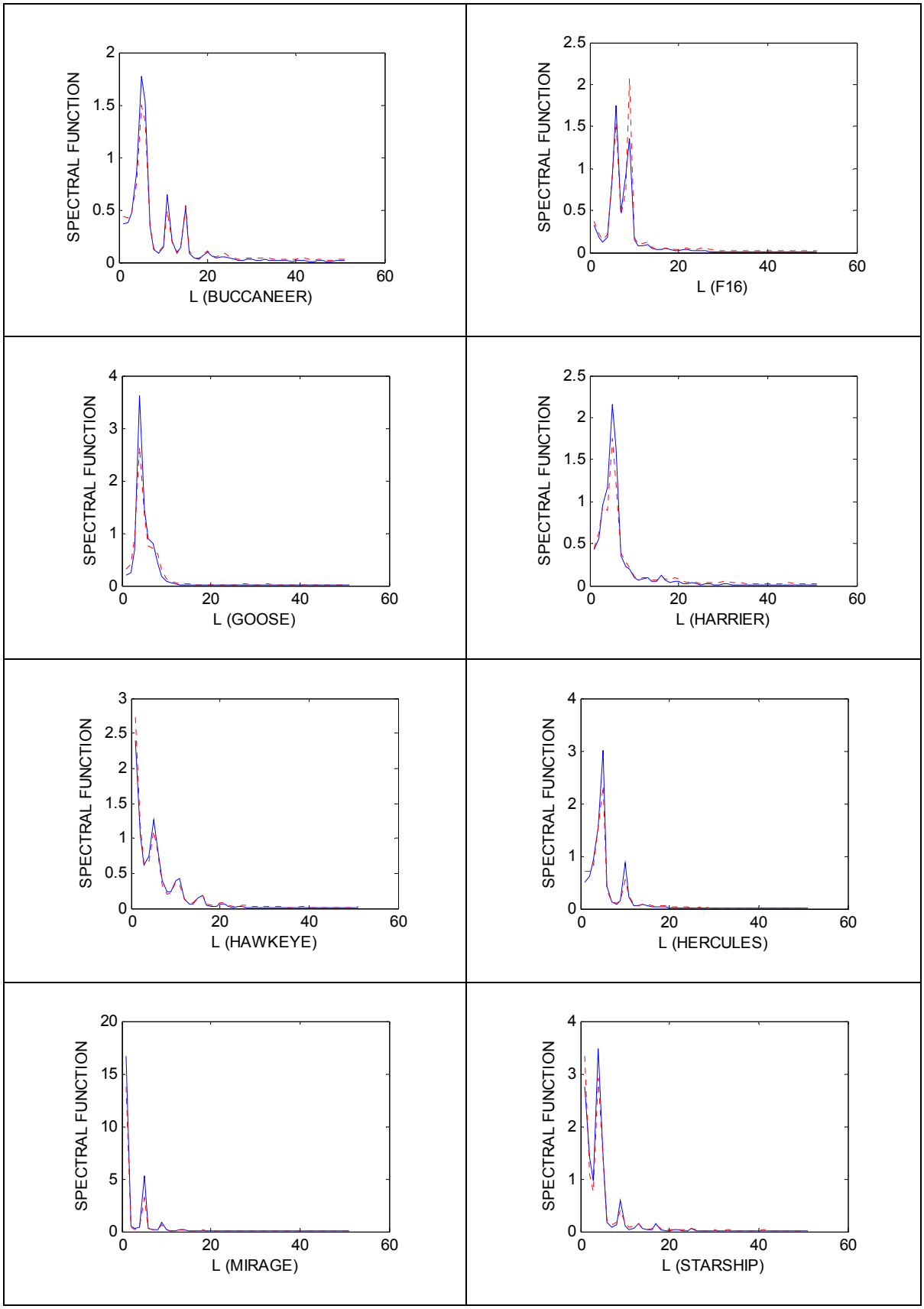


Fig. 15. Estimates of the spectral function $\tilde{f}_R$ for the templates in Fig. 8. The solid line gives the robust estimate in outlier free data, with the dashed line its value in outlier contaminated data ($s = 0.02$, $k = 12$). Here L refers to the frequency $2\pi L/100$.

Table 7
The percentage of correctly classified templates in Fig. 8 corrupted by noise with int(0.02T) outliers of magnitude k

		Robust			Non-robust		
Templates	**k**	***l*=50**	***l*=20**	**DG**	***l*=50**	***l*=20**	**DG**
Buccaneer	**0**	100	100	99	100	100	99
	12	*100*	*100*	*86*	*70*	*95*	*51*
F16	**0**	100	100	100	100	100	100
	12	*100*	*100*	*94*	*72*	*98*	*66*
Goose	**0**	100	100	100	100	100	100
	12	*99*	*100*	*100*	*42*	*85*	*72*
Harrier	**0**	100	100	99	100	100	100
	12	*96*	*100*	*84*	*85*	*93*	*55*
Hawkeye	**0**	95	93	100	100	100	97
	12	*94*	*100*	*82*	*55*	*84*	*60*
Hercules	**0**	98	99	100	100	100	99
	12	*86*	*100*	*95*	*40*	*75*	*53*
Mirage	**0**	100	100	100	100	100	100
	12	*98*	*99*	*86*	*68*	*98*	*61*
Starship	**0**	100	100	100	100	100	100
	12	*98*	*100*	*98*	*50*	*84*	*58*
Average	**0**	99.125	99	99.75	100	100	99.375
	12	*96.375*	*99.875*	*90.625*	*60.25*	*89*	*59.5*

Table 8
The percentage of correctly classified templates in Fig. 9 corrupted by noise with int(0.02T) outliers of magnitude k

		Robust			Non-robust		
Templates	**k**	***l*=50**	***l*=20**	**DG**	***l*=50**	***l*=20**	**DG**
Bar	**0**	100	100	100	100	100	100
	12	*97*	*99*	*94*	*80*	*82*	*96*
Bolt	**0**	90	96	100	96	95	99
	12	*68*	*92*	*93*	*54*	*53*	*50*
Boomerang	**0**	93	100	100	96	100	100
	12	*77*	*98*	*90*	*41*	*46*	*79*
Cylinder	**0**	100	100	100	100	100	100
	12	*100*	*100*	*100*	*100*	*100*	*100*
Nut	**0**	96	98	100	100	100	100
	12	*65*	*94*	*100*	*30*	*72*	*57*
Spanner	**0**	100	100	99	99	100	99
	12	*88*	*100*	*99*	*0*	*53*	*82*
Star	**0**	100	100	100	100	100	100
	12	*100*	*100*	*100*	*92*	*93*	*100*
Average	**0**	97	99.143	99.86	98.71	99.29	99.71
	12	*85*	*97.57*	*96.57*	*56.71*	*71.29*	*80.57*

Table 9
The percentage of correctly classified templates in Fig. 10 corrupted by noise with int(0.02T) outliers of magnitude k

		Robust			Non-robust		
Templates	**k**	***l*=50**	***l*=20**	**DG**	***l*=50**	***l*=20**	**DG**
G	**0**	78	97	100	79	98	100
	12	*85*	*95*	*100*	*86*	*96*	*92*
H	**0**	86	97	99	87	96	97
	12	*85*	*93*	*94*	*55*	*69*	*75*
I	**0**	99	100	100	100	100	100
	12	*76*	*93*	*97*	*57*	*59*	*90*
J	**0**	88	99	100	89	99	100
	12	*69*	*89*	*98*	*50*	*28*	*54*
Average	**0**	87.75	98.25	99.75	88.75	98.25	99.25
	12	*78.75*	*92.5*	*97.25*	*62*	*63*	*77.75*

Table 10
The confusion matrix for aircraft templates in the absence of outliers (k=0), using robust spectral approach (l=50) with its non-robust analogue in brackets

	Buccaneer	F16	Goose	Harrier	Hawkeye	Hercules	Mirage	Starship
Buccaneer	**100 (100)**	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
F16	0 (0)	**100 (100)**	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
Goose	0 (0)	0 (0)	**100 (100)**	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
Harrier	0 (0)	0 (0)	0 (0)	**100 (100)**	0 (0)	0 (0)	0 (0)	0 (0)
Hawkeye	0 (0)	0 (0)	0 (0)	0 (0)	**95 (100)**	5 (0)	0 (0)	0 (0)
Hercules	0 (0)	0 (0)	0 (0)	0 (0)	2 (0)	**98 (100)**	0 (0)	0 (0)
Mirage	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	**100 (100)**	0 (0)
Starship	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	**100 (100)**

Table 11
The confusion matrix for aircraft templates in the absence of outliers (k=0), using robust spectral approach (l=20) with its non-robust analogue in brackets

	Buccaneer	F16	Goose	Harrier	Hawkeye	Hercules	Mirage	Starship
Buccaneer	**100 (100)**	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
F16	0 (0)	**100 (100)**	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
Goose	0 (0)	0 (0)	**100 (100)**	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
Harrier	0 (0)	0 (0)	0 (0)	**100 (100)**	0 (0)	0 (0)	0 (0)	0 (0)
Hawkeye	0 (0)	0 (0)	0 (0)	0 (0)	**93 (100)**	7 (0)	0 (0)	0 (0)
Hercules	0 (0)	0 (0)	0 (0)	0 (0)	1 (0)	**99 (100)**	0 (0)	0 (0)
Mirage	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	**100 (100)**	0 (0)
Starship	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	**100 (100)**

Table 12
The confusion matrix for aircraft templates in the absence of outliers (k=0), using the robust Dubois and Glanz approach with its non-robust analogue in brackets

	Buccaneer	F16	Goose	Harrier	Hawkeye	Hercules	Mirage	Starship
Buccaneer	**99 (99)**	0 (0)	0 (0)	0 (0)	1 (1)	0 (0)	0 (0)	0 (0)
F16	0 (0)	**100 (100)**	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
Goose	0 (0)	0 (0)	**100 (100)**	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
Harrier	0 (0)	0 (0)	0 (0)	**99 (100)**	0 (0)	0 (0)	1 (0)	0 (0)
Hawkeye	0 (3)	0 (0)	0 (0)	0 (0)	**100 (97)**	0 (0)	0 (0)	0 (0)
Hercules	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	**100 (99)**	0 (0)	0 (1)
Mirage	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	**100 (100)**	0 (0)
Starship	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	**100 (100)**

Table 13
The confusion matrix for aircraft templates using robust spectral approach (l=50) with its non-robust analogue in brackets. In both cases the templates are corrupted by noise with int(0.02T) outliers of magnitude k = 12

	Buccaneer	F16	Goose	Harrier	Hawkeye	Hercules	Mirage	Starship
Buccaneer	**100 (70)**	0 (0)	0 (0)	0 (20)	0 (2)	0 (8)	0 (0)	0 (0)
F16	0 (0)	**100 (72)**	0 (1)	0 (22)	0 (2)	0 (3)	0 (0)	0 (0)
Goose	0 (0)	0 (0)	**99 (42)**	1 (52)	0 (0)	0 (6)	0 (0)	0 (0)
Harrier	0 (0)	0 (0)	0 (8)	**96 (85)**	0 (2)	4 (5)	0 (0)	0 (0)
Hawkeye	0 (0)	0 (0)	0 (0)	0 (40)	**94 (55)**	1 (1)	5 (4)	0 (0)
Hercules	4 (1)	0 (0)	0 (0)	10 (59)	0 (0)	**86 (40)**	0 (0)	0 (0)
Mirage	0 (0)	0 (0)	0 (6)	0 (23)	2 (3)	0 (0)	**98 (68)**	0 (0)
Starship	0 (0)	0 (0)	0 (0)	0 (46)	0 (0)	0 (4)	2	**98 (50)**

Table 14
The confusion matrix for aircraft templates using robust spectral approach (l=20) with its non-robust analogue in brackets. In both cases the templates are corrupted by noise with int(0.02T) outliers of magnitude k = 12

	Buccaneer	F16	Goose	Harrier	Hawkeye	Hercules	Mirage	Starship
Buccaneer	**100 (95)**	0 (0)	0 (0)	0 (4)	0 (0)	0 (0)	0 (1)	0 (0)
F16	0 (0)	**100 (98)**	0 (0)	0 (2)	0 (0)	0 (0)	0 (0)	0 (0)
Goose	0 (0)	0 (0)	**100 (85)**	0 (12)	0 (3)	0 (0)	0 (0)	0 (0)
Harrier	0 (0)	0 (0)	0 (6)	**100 (93)**	0 (1)	0 (0)	0 (0)	0 (0)
Hawkeye	0 (0)	0 (0)	0 (0)	0 (8)	**100 (84)**	0 (0)	0 (8)	0 (0)
Hercules	0 (0)	0 (0)	0 (25)	0 (0)	0 (0)	**100 (75)**	0 (0)	0 (0)
Mirage	0 (0)	0 (0)	0 (0)	0 (2)	1 (0)	0 (0)	**99 (98)**	0 (0)
Starship	0 (0)	0 (0)	0 (1)	0 (13)	0 (1)	0 (0)	0 (1)	**100 (84)**

Table 15
The confusion matrix for aircraft templates using the robust Dubois and Glanz approach with its non-robust analogue in brackets. In both cases the templates are corrupted by noise with int(0.02T) outliers of magnitude k = 12

	Buccaneer	F16	Goose	Harrier	Hawkeye	Hercules	Mirage	Starship
Buccaneer	**86 (51)**	1 (4)	0 (4)	1 (18)	4 (14)	0 (0)	8 (7)	0 (2)
F16	5 (8)	**94 (66)**	0 (4)	0 (8)	0 (0)	0 (0)	1 (14)	0 (0)
Goose	0 (1)	0 (1)	**100 (72)**	0 (12)	0 (3)	0 (7)	0 (1)	0 (3)
Harrier	3 (14)	0 (2)	0 (8)	**84 (55)**	8 (5)	0 (4)	5 (5)	0 (7)
Hawkeye	8 (13)	0 (7)	0 (3)	7 (10)	**82 (60)**	0 (1)	3 (0)	0 (6)
Hercules	1 (0)	0 (1)	2 (13)	0 (4)	0 (8)	**95 (53)**	0 (0)	2 (21)
Mirage	4 (4)	2 (13)	0 (5)	6 (15)	2 (2)	0 (0)	**86 (61)**	0 (0)
Starship	0 (0)	0 (0)	0 (8)	1 (7)	0 (10)	0 (17)	1 (0)	**98 (58)**

Table 16
The confusion matrix for machine part templates in the absence of outliers (k=0), using robust spectral approach (l=50) with its non-robust analogue in brackets

	Bar	Bolt	Boomerang	Cylinder	Nut	Spanner	Star
Bar	**100 (100)**	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
Bolt	0 (0)	**90 (96)**	2 (0)	0 (0)	3 (0)	5 (4)	0 (0)
Boomerang	0 (0)	0 (0)	**93 (96)**	0 (0)	2 (1)	5 (3)	0 (0)
Cylinder	0 (0)	0 (0)	0 (0)	**100 (100)**	0 (0)	0 (0)	0 (0)
Nut	0 (0)	4 (0)	0 (0)	0 (0)	**96 (100)**	0 (0)	0 (0)
Spanner	0 (0)	0 (1)	0 (0)	0 (0)	0 (0)	**100 (99)**	0 (0)
Star	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	**100 (100)**

Table 17
The confusion matrix for machine part templates in the absence of outliers (k=0), using robust spectral approach (l=20) with its non-robust analogue in brackets

	Bar	Bolt	Boomerang	Cylinder	Nut	Spanner	Star
Bar	**100 (100)**	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
Bolt	0 (0)	**96 (95)**	0 (0)	0 (0)	0 (0)	4 (5)	0 (0)
Boomerang	0 (0)	0 (0)	**100 (100)**	0 (0)	0 (0)	0 (0)	0 (0)
Cylinder	0 (0)	0 (0)	0 (0)	**100 (100)**	0 (0)	0 (0)	0 (0)
Nut	0 (0)	2 (0)	0 (0)	0 (0)	**98 (100)**	0 (0)	0 (0)
Spanner	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	**100 (100)**	0 (0)
Star	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	**100 (100)**

Table 18
The confusion matrix for machine part templates in the absence of outliers (k=0), using the robust Dubois and Glanz approach with its non-robust analogue in brackets

	Bar	Bolt	Boomerang	Cylinder	Nut	Spanner	Star
Bar	**100 (100)**	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
Bolt	0 (0)	**100 (99)**	0 (0)	0 (0)	0 (0)	0 (1)	0 (0)
Boomerang	0 (0)	0 (0)	**100 (100)**	0 (0)	0 (0)	0 (0)	0 (0)
Cylinder	0 (0)	0 (0)	0 (0)	**100 (100)**	0 (0)	0 (0)	0 (0)
Nut	0 (0)	0 (0)	0 (0)	0 (0)	**100 (100)**	0 (0)	0 (0)
Spanner	0 (0)	1 (1)	0 (0)	0 (0)	0 (0)	**99 (99)**	0 (0)
Star	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	**100 (100)**

Table 19
The confusion matrix for machine part templates using robust spectral approach (l=50) with its non-robust analogue in brackets. In both cases the templates are corrupted by noise with int(0.02T) outliers of magnitude k = 12

	Bar	Bolt	Boomerang	Cylinder	Nut	Spanner	Star
Bar	**97 (80)**	0 (0)	0 (6)	3 (12)	0 (2)	0 (0)	0 (0)
Bolt	6 (46)	**68 (54)**	0 (0)	0 (0)	19 (0)	7 (0)	0 (0)
Boomerang	3 (52)	4 (7)	**77 (41)**	0 (0)	16 (0)	0 (0)	0 (0)
Cylinder	0 (0)	0 (0)	0 (0)	**100 (100)**	0 (0)	0 (0)	0 (0)
Nut	14 (41)	13 (4)	8 (25)	0 (0)	**65 (30)**	0 (0)	0 (0)
Spanner	2 (54)	10 (35)	0 (9)	0 (1)	0 (1)	**88 (0)**	0 (0)
Star	0 (5)	0 (0)	0 (0)	0 (3)	0 (0)	0 (0)	**100 (92)**

Table 20
The confusion matrix for machine part templates using robust spectral approach (l=20) with its non-robust analogue in brackets. In both cases the templates are corrupted by noise with int(0.02T) outliers of magnitude k = 12

	Bar	Bolt	Boomerang	Cylinder	Nut	Spanner	Star
Bar	**99 (82)**	0 (0)	0 (10)	1 (8)	0 (0)	0 (0)	0 (0)
Bolt	0 (31)	**92 (53)**	2 (0)	0 (0)	0 (1)	6 (15)	0 (0)
Boomerang	0 (16)	2 (0)	**98 (46)**	0 (0)	0 (38)	0 (0)	0 (0)
Cylinder	0 (0)	0 (0)	0 (0)	**100 (100)**	0 (0)	0 (0)	0 (0)
Nut	4 (18)	0 (0)	2 (10)	0 (0)	**94 (72)**	0 (0)	0 (0)
Spanner	0 (24)	0 (13)	0 (0)	0 (0)	0 (10)	**100 (53)**	0 (0)
Star	0 (7)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	**100 (93)**

Table 21
The confusion matrix for machine part templates using the robust Dubois and Glanz approach with its non-robust analogue in brackets. In both cases the templates are corrupted by noise with int(0.02T) outliers of magnitude k = 12

	Bar	Bolt	Boomerang	Cylinder	Nut	Spanner	Star
Bar	**94 (96)**	0 (0)	0 (0)	6 (4)	0 (0)	0 (0)	0 (0)
Bolt	0 (0)	**93 (50)**	6 (3)	0 (0)	1 (18)	0 (29)	0 (0)
Boomerang	0 (0)	7 (1)	**90 (79)**	0 (0)	3 (4)	0 (16)	0 (0)
Cylinder	0 (0)	0 (0)	0 (0)	**100 (100)**	0 (0)	0 (0)	0 (0)
Nut	0 (0)	0 (30)	0 (5)	0 (0)	**100 (57)**	0 (8)	0 (0)
Spanner	0 (0)	1 (4)	0 (9)	0 (0)	0 (5)	**99 (82)**	0 (0)
Star	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	**100 (100)**

Table 22
The confusion matrix for letter templates in the absence of outliers (k=0), using robust spectral approach (l=50) with its non-robust analogue in brackets

	G	H	I	J
G	**78 (79)**	20 (19)	0 (0)	2 (2)
H	10 (9)	**86 (87)**	0 (0)	4 (4)
I	0 (0)	0 (0)	**99 (100)**	1 (0)
J	0 (0)	10 (11)	2	**88 (89)**

Table 23
The confusion matrix for letter templates in the absence of outliers (k=0), using robust spectral approach (l=20) with its non-robust analogue in brackets

	G	H	I	J
G	**97 (98)**	2 (1)	0 (0)	1 (1)
H	2 (3)	**97 (96)**	0 (0)	1 (1)
I	0 (0)	0 (0)	**100 (100)**	0 (0)
J	0 (0)	1 (1)	0 (0)	**99 (99)**

Table 24
The confusion matrix for letter templates in the absence of outliers (k=0), using the robust Dubois and Glanz approach with its non-robust analogue in brackets

	G	H	I	J
G	**100 (100)**	0 (0)	0 (0)	0 (0)
H	0 (0)	**99 (97)**	0 (0)	1 (3)
I	0 (0)	0 (0)	**100 (100)**	0 (0)
J	0 (0)	0 (0)	0 (0)	**100 (100)**

Table 25
The confusion matrix for letter templates using robust spectral approach (l=50) with its non-robust analogue in brackets. In both cases the templates are corrupted by noise with int(0.02T) outliers of magnitude k = 12

	G	H	I	J
G	**85 (86)**	14 (3)	0 (1)	1 (10)
H	6 (22)	**85 (55)**	0 (0)	9 (23)
I	7 (30)	3 (8)	**76 (57)**	14 (5)
J	4 (24)	25 (26)	2 (0)	**69 (50)**

Table 26
The confusion matrix for letter templates using robust spectral approach (l=20) with its non-robust analogue in brackets. In both cases the templates are corrupted by noise with int(0.02T) outliers of magnitude k = 12

	G	H	I	J
G	**95 (96)**	3 (4)	0 (0)	2 (0)
H	0 (2)	**93 (69)**	0 (5)	7 (24)
I	0 (1)	0 (9)	**93 (59)**	7 (31)
J	0 (0)	9 (53)	2 (19)	**89 (28)**

Table 27
The confusion matrix for letter templates using the robust Dubois and Glanz approach with its non-robust analogue in brackets. In both cases the templates are corrupted by noise with int(0.02T) outliers of magnitude k = 12

	G	H	I	J
G	**100 (92)**	0 (1)	0 (1)	0 (6)
H	0 (10)	**94 (75)**	0 (0)	6 (15)
I	0 (0)	0 (0)	**97 (90)**	3 (10)
J	0 (12)	2 (33)	0 (1)	**98 (54)**

In order to examine the effect of lag selection procedure and use of sub-set auto regression instead of full auto regression, we generate training and test data for aircraft templates using full CAR model of order 5. We consider the outlier contaminated data (s=0.02, k=12) and take l = 20. The obtained confusion matrix is given in table 28. By comparing tables 28 and 14, we see that the performance reduces significantly when we consider full model parameter estimates. This test shows that selecting near optimum lag structure for each template significantly improves the recognition performance.

Table 28
The confusion matrix for aircraft templates using robust spectral approach (l=20) based on parameter estimates of full CAR model of order 5. Templates are corrupted by noise with int(0.02T) outliers of magnitude k = 12

	Buccaneer	F16	Goose	Harrier	Hawkeye	Hercules	Mirage	Starship
Buccaneer	**63**	2	0	0	8	0	27	0
F16	9	**89**	0	0	0	0	2	0
Goose	0	0	**94**	0	0	6	0	0
Harrier	2	0	2	**48**	9	31	5	3
Hawkeye	9	0	0	4	**77**	3	7	0
Hercules	0	0	1	18	1	**75**	1	4
Mirage	22	10	1	8	9	12	**38**	0
Starship	0	0	0	0	0	15	0	**85**

- Average correct classification percentage = 71.125%

As we stated before, in our experiment we did not use the lag selection procedure in the test set. However, in order to see the effect of this procedure we do a sample test here. We consider the robust spectral classifier for aircraft templates in the presence of outliers and perform the classification task employing the lag selection procedure in the test set. It is noted that in the case of test data, lag selection must be done independently for each record, so in section 4.3 we must take $N = 1$ in all equations. Table 29 presents the result. By comparing tables 29 and 14, we see that in addition to significant speed reduction, the performance is also reduces when we perform lag selection in the test set. It is shown in [1] that the use of lag selection in the test set can improve the classification sensitivity to 'clutters' (objects not in the training set).

Table 29
The confusion matrix for aircraft templates using robust spectral approach (l=20) and eploying lag selection procedure in test set. Templates are corrupted by noise with int(0.02T) outliers of magnitude k = 12

	Buccaneer	F16	Goose	Harrier	Hawkeye	Hercules	Mirage	Starship
Buccaneer	**89**	0	0	7	6	5	0	0
F16	1	**98**	0	0	0	0	1	0
Goose	0	0	**97**	2	0	1	0	0
Harrier	3	0	2	**83**	0	12	0	0
Hawkeye	1	0	0	1	**57**	41	0	0
Hercules	0	0	1	17	6	**73**	3	0
Mirage	0	1	0	1	7	0	**91**	0
Starship	0	0	0	1	2	1	0	**96**

- Average correct classification percentage = 84.625%

To evaluate the performance of the spectral classifier in recognizing deformed shapes, we test it through the same problem that we described in section 3.4. We consider the data before corruption (the data without noise and outliers) and estimates training and test data. In this case we use the lag structures obtained for 'Arial' in the absence of outliers and use them for test sets ('Book Antiqua', 'Times New Roman', and 'Tahoma') too. We employ robust procedures, although in this case (in the absence of outliers) their results are not very different from those of non-robust procedures.

Tables 30, 31, and 32 summarize the obtained results in our experiment. Again we see that the results are not satisfactory and even our new classifier based on spectral function estimates can not be applied to this problem.

Table 30
The confusion matrix for 'Book Antiqua' before data corruption using robust spectral approach (l=20).

	G	H	I	J
G	**0**	14	0	86
H	9	**90**	0	1
I	0	73	**1**	26
J	0	9	4	**87**

Average correct classification percentage = 44.5%

Table 31
The confusion matrix for 'Times New Roman' before data corruption using robust spectral approach (l=20).

	G	H	I	J
G	**1**	72	0	27
H	26	**74**	0	0
I	0	58	**0**	42
J	0	1	0	**99**

Average correct classification percentage = 43.5%

Table 32
The confusion matrix for 'Tahoma' before data corruption using robust spectral approach (l=20).

	G	H	I	J
G	**90**	10	0	0
H	0	**87**	0	13
I	9	91	**0**	0
J	0	99	0	**1**

Average correct classification percentage = 44.5%

5. Conclusions

In This project we considered the problem of classifying objects using two dimensional boundary data in both noise free environment and environment generating significant imaging noise.

The later increasing the likelihood of contamination by large aberrant observations (outliers). The shape classification systems were mainly based on the CAR model parameter representation of two-dimensional shape boundaries. In order to obtain the boundary samples from which the CAR model parameters were estimated, a boundary approximation scheme was developed to determine the lengths of N equiangularly spaced radius vectors projected between the boundary centroid and the boundary. This scheme accurately represents convex shapes and complicated concave shapes. Because of the properties of the boundary approximation scheme and the CAR model itself, the parameters of the CAR model are approximately invariant to shape size, and translational and rotational position. Two feature based pattern recognition schemes were implemented and studied in noise-free environment. In the contaminated environment, circular sub-set auto regressions with robust lag selection and parameter estimation were used to estimate the spectral function associated with object boundaries and classify unknown templates using a 'distance' based classifier. The suggested robust approach substantially outperforms non-robust techniques (based on the sample covariance function) which suffer catastrophic reductions in performance in outlier contaminated data. It was shown that the robust lag selection procedure is quite advantageous for a wide range of templates in contaminated environment.

In our experiment we did not use the lag selection procedure in the test set because of the significant speed reduction. However it can be shown, see [1], that performing lag selection in the test set, adapts the model structure to the data and is well suited to classification problems where sensitivity to clutter is important.

According to our results we can generally say that the CAR model parameters are useful shape descriptors for recognition purposes. We obtained successful classification results for a wide range of convex and concave shapes in various sizes and spatial positions by the direct use of these parameters or using them for estimating spectral functions. In noise free environment we saw that the model order of each individual shape sample does not have to be determined for accurate recognition and shapes can be successfully classified by the use of CAR models of order lower than optimum. However the lag selection procedure introduced in section 4.3 can be used in problems that are sensitive to the model order.

We did also some testing of the performance of our algorithms on deformed shapes. Although our results were not very satisfactory, the CAR model parameters can be potentially used for these problems too and further studies may lead to much better results. It is noted that there is a tradeoff between sensitivity to the main shapes and insensitivity to deformation and this tradeoff is implicit in any effort to recognize deformed shapes.

References

[1] R.H. Glendinning, "Robust shape classification," *Signal Processing, vol. 77, pp. 121 – 138, 1999.*

[2] S. R. Dubois and F. H. Glanz, "An Autoregressive Model Approach to Two-Dimensional Shape Classification," *IEEE Trans. Pattern Anal. Mach. Intell., vol. PAMI-8, pp. 55 – 66, 1986.*

[3] R. L. Kashyap and R. Chellappa, "Stochastic models for closed boundary analysis: Representation and reconstruction," *IEEE Trans. Inform. Theory, vol. IT-27, pp. 627 – 637, 1981.*

[4] R. O. Duda, P. E. Hart, and D. G. Stork, "Pattern Classification," *2nd Ed., Wiley, 2000.*

Appendix

MATLAB Programs

1. Boundary Extraction

BoundaryExtraction.m

```
function [Boundary]=BoundaryExtraction(Image)  % Image: White Object in Black Background
Image=ImClean(Image);
[r c]=find(Image==1);
[p q]=min(r);
Boundary(1,:)=[p(1) c(q(1))];
CurrentPoint=Boundary(1,:)+1;
Direction=[0 1;1 1;1 0;1 -1;0 -1; -1 -1;-1 0;-1 1];
i=1;
DirNo=1;
[redacted]
            DirNo=1;
        else
            DirNo=DirNo+1;
        end
    end
    Boundary(i+1,:)=Boundary(i,:)+Direction(DirNo,:);
  else
    while Image( (Boundary(i,1)+Direction(DirNo,1)),(Boundary(i,2)+Direction(DirNo,2)) )==1
      Boundary(i+1,:)=Boundary(i,:)+Direction(DirNo,:);
      if DirNo==1;
          DirNo=8;
      else
          DirNo=DirNo-1;
      end
    end
  end
  i=i+1;
  CurrentPoint=Boundary(i,:);
end
Boundary(end,:)=[];
```

Imclean.m

```
function [CleanImage]=ImClean(Image)
LUT = makelut('sum(x(:)) <=3',3);
Image =~(applylut(Image,LUT));
[redacted]
```

2. Boundary Sampling

TimeSeries.m

```
function [r_t]=TimeSeries(Boundary,AngleNo)
%function [r_t, Intersections]=TimeSeries(Boundary,AngleNo)
Centroid=round(mean(Boundary));
N=[0:fix(AngleNo/4)-1];
Slopes=tan(2*pi*N/AngleNo);
MaxSlopeNo=size(Slopes,2)+1;
Slopes(MaxSlopeNo)=10000; % a large value for tan(pi/2)
OldSlopeNo=MaxSlopeNo+1;
r_t=[];
counter=0;
%Intersections=[Centroid(1) Centroid(2)];
i=1;
InitializationFlag=0;
[redacted]
                =SlopeNo0+1;
                RSD0=s-Slopes(SlopeNo0);
                RSD1=s-Slopes(SlopeNo1);
                if OldSlopeNo~=SlopeNo1
                    r_t(end+1)=norm(Boundary(i,:)-Centroid);
                    OldSlopeNo=SlopeNo1;
%                     Intersections(end+1,:)=Boundary(i,:);
                end
            else if (SD1*RSD1)<0
                    m=find(Slopes <= s);
                    SlopeNo0=m(end);
                    SlopeNo1=SlopeNo0+1;
                    RSD0=s-Slopes(SlopeNo0);
                    RSD1=s-Slopes(SlopeNo1);
                    if OldSlopeNo~=SlopeNo0
                        r_t(end+1)=norm(Boundary(i,:)-Centroid);
                        OldSlopeNo=SlopeNo0;
%                         Intersections(end+1,:)=Boundary(i,:);
                    end
                end
            end
        end
    end
  end
  i=i+1;
end
```

3. Boundary Modeling

3.1. Circular Full Auto Regressive Model of order *m*

- The formulas presented in section 2.3 are used in this program.
- This program was used in noise-free environment (section 3).
- Instead of this program we can use the next program (*CSARModel.m*) choosing $J = (1, 2, \ldots, m)$.

CFARModel.m

```
function [Theta, Alpha, Beta]=CFARModel(r_t, M) % M = Model Order
N=size(r_t,2);
for i=1:M
    Zt_1(i,:)=circshift(r_t,[0 i]);
[redacted]
                                    (1:M);
Alpha=ThetaAlpha(end);
Beta=sum( (r_t - ThetaAlpha' *Ut_1).^2 ) /N;
```

3.2. Circular Sub-set Auto Regressive Model

- The Yule-Walker equation is used in this program with non-robust estimations (based on sample covariance function).
- This program was used in section 4 wherever non-robust procedures were desired.

CSARModel.m

```
function [Phi_hat,Sigma2_hat]=CSARModel(Y, J)
T=size(Y,2);
M=size(J,1);
r_hat0=Autocovariance(Y,0);
R_hat=r_hat0*eye(M);
for i=1:M
    r_hat(i,1)=Autocovariance(Y,J(i));
    for j=i+1:M
        R_hat(i,j)=Autocovariance( Y, abs(J(i)-J(j)) );
    end
end
R_hat=R_hat+(triu(R_hat,1))';
[redacted]

[redacted]
Myu=mean(Y);
N=size(Y,2);
Ac=( Y - Myu )*( circshift(Y,[0 Lag]) - Myu )'/N ;
```

3.3. Robust Circular Sub-set Auto Regressive Model

- The Yule-Walker equation is used in this program with robust estimations given by Eqs. (9), (10), (11), and (12).
- This program was used in section 4 wherever robust procedures were desired.

RobustCSARModel.m

```
function [Phi_hat,Sigma2_hat]=RobustCSARModel(Y, J, f)
T=size(Y,2);
M=size(J,1);
Myu_hat=median(Y);
SigmaX_hat=1.483*median(abs(Y-Myu_hat));
r_hat0=RobustAc(Y, Myu_hat, SigmaX_hat, 0, f);
R_hat=r_hat0*eye(M);
for i=1:M
[redacted]
end
R_hat=R_hat+(triu(R_hat,1))';
Phi_hat = inv(R_hat)*r_hat ;
Sigma2_hat=(T-M)/(T-2*M-1)*(r_hat0-sum(Phi_hat.*r_hat));

[redacted]
c_hat=Psi*( circshift(Psi,[0 Lag]) )'/T;
c_hat0=Psi*Psi'/T ;
RAc= SigmaX_hat^2 * c_hat/c_hat0;
```

4. Lag Selection

LagSelection.m

```
load Letters\Arial\Data\Noisy\TrainSeries
load Letters\Arial\Data\HubersParameter

M_Init=20;  % Initial Model Order
Beta=0.2;   % Balance factor in Loss Function
J=[];
[redacted];
   J(1:size(RLags,1),end+1)=RLags; % since lag structuers have different sizes, gaps are filled by 0
   M(i)=size(RLags,1); % Store the size of each lag structure
   i
end
% save('Letters\Arial\Data\Noisy\RobustLags','J','M');
```

RobustLags.m

```
function RLags=RobustLags(Y,T,f,M,Beta)
Gama=1;
Jv=([1:M])';
CandidateLags=Jv;
N=size(Y,1);
for v=1:M-1
   for k=1:N
      if v==1
         Yk=Y(k,1:T(k));
         SigmaX_hat(k)=1.483*median(abs(Yk-median(Yk)));
      end
      Yk=Y(k,1:T(k));
      [Phi(:,k) SigmaZ2(k)]=RobustCSARModel(Yk/SigmaX_hat(k),Jv,f);
[redacted]
      SigmaZ2_tilda(v)=SigmaZ2;
```

```
    LossFunction=Beta*log10(Phi.^2);
  end
  [u j]=min(LossFunction);
  Jv(j(1))=[];
  CandidateLags(1:M-v,end+1)=Jv;
  clear Phi;
  v
end
RFPE=SigmaZ2_tilda.*( 1 + Gama*([M:-1:2]+1)/sum(T) );
[u j]=min(RFPE);
m=find(CandidateLags(:,j)>0);
RLags=CandidateLags(1:m(end),j);
```

5. Training and Test data generation

5.1. Generating Training and Test series

- This program generates clean Training and Test time series.

TSGenerator.m

```
Image(:,:,1)=~imread('Letters\BookAntiqua\G','bmp');
Image(:,:,2)=~imread('Letters\BookAntiqua\H','bmp');
Image(:,:,3)=~imread('Letters\BookAntiqua\I','bmp');
Image(:,:,4)=~imread('Letters\BookAntiqua\J','bmp');
Ntrain=100;     % Number of training data
AngleNo=76;     % Number of angles for sampeling(it must be a multiple of 4)
ScaleFactor=round(unifrnd(70,130,[1 Ntrain]))/100;  %Images are scaled from 70% up to 130%
RotationAngles=randperm(360);          %Images are rotated from 0d up to 360d. For Ntrain > 360 another...
RotationAngles=RotationAngles(1:Ntrain);% ... random generator function must be used

C=size(Image,3);    % Number of classes
Xt=[];
for i=1:Ntrain
   ScaledIm=imresize(Image,ScaleFactor(i));
   RotatedScaledIm=imrotate(ScaledIm,RotationAngles(i),'bicubic');
   [redacted]
      T(i,j)=size(r_t,2);
      j
   end
   i
end
% save('Letters\BookAntiqua\Data\Clean\TestSeries','Xt','T')
% save('Letters\BookAntiqua\Data\Clean\SigmaX','SigmaX') % SigmaX may not be needed
```

- This program adds contamination to the clean time series.

DataContaminator.m

```
load Letters\Arial\Data\Clean\TrainSeries
%load Letters\Arial\Data\Clean\TestSeries

SigmaN=4;
k=12;
s=0.02;
% Notice! comment the specified lines when use this program for generating test data
NoisyData=zeros(size(Xt));
ContaminatedData=zeros(size(Xt));
for i=1:size(Xt,1)
   for j=1:size(Xt,3)
      X=Xt(i,1:T(i,j),j);
      [redacted]
```

```
      [redacted]
      Myu_hat=median(Y);                    % Comment
      SigmaX_hat=1.483*median(abs(Y-Myu_hat));    % Comment
      F(i,j)=max(abs((X-Myu_hat)/SigmaX_hat));    % Comment

      A=[zeros(1,T(i,j)-fix(s*T(i,j))) sign(unifrnd(-1,1,[1 fix(s*T(i,j))]))];
      A=A(randperm(T(i,j)));
      SigmaX=sqrt(var(X));
      Y=X + N + k*SigmaX*A;
      m=find(Y<0);
      Y(m)=0;
      ContaminatedData(i,1:T(i,j),j)=Y;
   end
end
% Yt=NoisyData;
% save('Letters\Arial\Data\Noisy\TrainSeries','Yt','T');
% Yt=ContaminatedData;
% save('Letters\Arial\Data\Contaminated\TrainSeries','Yt','T');
% F=max(F);                                 % Comment
% save('Letters\Arial\Data\HubersParameter','F');     % Comment

% Y1=Xt(i,1:T(i,j),j);
% Y2=NoisyData(i,1:T(i,j),j);
% Y3=ContaminatedData(i,1:T(i,j),j);
% stairs(Y1)
% figure,stairs(Y2)
% figure,stairs(Y3)
```

5.2. Generating data for Dubois and Glanz approach

- This program generates Training or Test data for noise-free environment using *CFARModel.m*.
- This program was used in section 3.

TrainDataGenerator.m

```
Image(:,:,1)=~imread('Letters\BookAntiqua\G','bmp');
Image(:,:,2)=~imread('Letters\BookAntiqua\H','bmp');
Image(:,:,3)=~imread('Letters\BookAntiqua\I','bmp');
Image(:,:,4)=~imread('Letters\BookAntiqua\J','bmp');
Ntrain=25;  % Number of training data
M=2;        % Model Order
[redacted]
RotationAngles=randperm(360);          %Images are rotated from 0d up to 360d. For Ntrain > 360 another...
RotationAngles=RotationAngles(1:Ntrain);% ... random generator function must be used

C=size(Image,3);    % Number of classes
for i=1:Ntrain
   [redacted]
      Xtrain02(:,i,j)=[Theta; Alpha/sqrt(Beta)];
      [Theta, Alpha, Beta]=CARModel(r_t , 3);
      Xtrain03(:,i,j)=[Theta; Alpha/sqrt(Beta)];
      [Theta, Alpha, Beta]=CARModel(r_t , 4);
      Xtrain04(:,i,j)=[Theta; Alpha/sqrt(Beta)];
      [Theta, Alpha, Beta]=CARModel(r_t , 5);
      Xtrain05(:,i,j)=[Theta; Alpha/sqrt(Beta)];
      [Theta, Alpha, Beta]=CARModel(r_t , 6);
      Xtrain06(:,i,j)=[Theta; Alpha/sqrt(Beta)];
      [Theta, Alpha, Beta]=CARModel(r_t , 7);
      Xtrain07(:,i,j)=[Theta; Alpha/sqrt(Beta)];
      [Theta, Alpha, Beta]=CARModel(r_t , 8);
      Xtrain08(:,i,j)=[Theta; Alpha/sqrt(Beta)];
      [Theta, Alpha, Beta]=CARModel(r_t , 9);
      Xtrain09(:,i,j)=[Theta; Alpha/sqrt(Beta)];
      [Theta, Alpha, Beta]=CARModel(r_t , 10);
```

```
        Xtrain10(:,i,j)=[Theta; Alpha/sqrt(Beta)];
        j
    end
    i
end
```

- This program generates Training and Test data for contaminated environment using *CSARModel.m* and *RobustCSARModel.m*
- The robust part must be commented when non-robust data is desired.

DGDataGenerator.m

```
load Letters\Arial\Data\Contaminated\TrainSeries
% load Letters\Arial\Data\Contaminated\TestSeries
load Letters\Arial\Data\HubersParameter
M=5;    % Full Model Order

% Notice! The robust part must be commented when non-robust data is desired
J=([1:M])';
for i=1:size(Yt,1)
    for j=1:size(Yt,3)
        Y=Yt(i,1:T(i,j),j);

        % Non-Robust ----------------------------------
%       [Phi_hat SigmaZ2_hat]=CSARModel(Y,J);
[redacted]
        Myu_hat=median(Y);

        X(:,i,j)=[Phi_hat ; Myu_hat/sqrt(SigmaZ2_hat)];
    end
end
% Xtrain=X;
% save('Letters\Arial\Data\Contaminated\RobustDGXtrain','Xtrain')

% Xtest=X;
% save('Letters\Arial\Data\Contaminated\RobustDGXtest','Xtest')
```

5.3. Generating data for Spectral function approach

- This program generates Training data for spectral approach using *CSARModel.m* and *RobustCSARModel.m*
- The robust part must be commented when non-robust data is desired.

SFArgumentsEstimator.m

```
load Aircrafts\Data\Contaminated\TrainSeries
load Aircrafts\Data\Contaminated\RobustLags
load Aircrafts\Data\HubersParameter

Phi_tilda=[];
SigmaZ2_tilda=[];
% Notice! The robust part must be commented when non-robust data is desired
for j=1:size(Yt,3);
[redacted]
        [Phi(:,i) SigmaZ2(i)]=CSARModel(Y/SigmaX_hat,Jj);
        % Robust ------------------------------------------------
%       SigmaX_hat=1.483*median(abs(Y-median(Y)));
%       [Phi(:,i) SigmaZ2(i)]=RobustCSARModel(Y/SigmaX_hat,Jj,F(j));

    end
    Phi_tilda(1:size(Phi,1),j)=mean(Phi,2);
```

```
    SigmaZ2_tilda(j)=mean(SigmaZ2);
    clear Phi
    j
end
% save('Aircrafts\Data\Contaminated\NonRobustTrainSFArgs','Phi_tilda','SigmaZ2_tilda');
```

- This program generates Test data for spectral approach using *CSARModel.m* and *RobustCSARModel.m*
- The robust part must be commented when non-robust data is desired.

SFArgumentsEstimator_Test.m

```
load Aircrafts\Data\Contaminated\TestSeries
load Aircrafts\Data\Contaminated\RobustLags
load Aircrafts\Data\HubersParameter

Phi_hat=[];
SigmaZ2_hat=[];
% Notice! The robust part must be commented when non-robust data is desired
for j=1:size(Yt,3);
    [redacted]

        [redacted] Non-robust ----------------------------------------------------
        SigmaX_hat=sqrt(var(Y));
        [Phi_hat(1:M(j),i,j) SigmaZ2_hat(i,j)]=CSARModel(Y/SigmaX_hat,Jj);
        % Robust -------------------------------------------------------
%       SigmaX_hat=1.483*median(abs(Y-median(Y)));
%       [Phi_hat(1:M(j),i,j) SigmaZ2_hat(i,j)]=RobustCSARModel(Y/SigmaX_hat,Jj,F(j));

    end
    j
end
% save('Aircrafts\Data\Contaminated\NonRobustTestSFArgs','Phi_hat','SigmaZ2_hat','T');
```

6. Classifiers

6.1. Bayes optimal classifier with Gaussian parametric estimation of PDFs

BayesClassifier.m

```
% Minimm error Bayes classifier----------------------------------------------------------------
load Letters\Arial\Data\Contaminated\RobustDGXtrain
load Letters\Arial\Data\Contaminated\RobustDGXtest
%Xtrain(:,21:end,:)=[]; % Reducing the size of training set

[L Ntrain C]=size(Xtrain);
Ntest=size(Xtest,2);
Pw(1:C)=1/C;    % a priori class distr bution

[Myu,Sigma]=MLPdfEstimator(Xtrain);
[redacted]
for k=1:Ntest
    for i=1:C
        for j=1:C
            difference=Xtest(:,k,i)-Myu(:,1,j);
            f_xk_w(j)=exp(-0.5*difference' * SigmaInv(:,:,j) * difference) / const(j);
        end
        [m,J]=max(Pw.*f_xk_w);
        Confusion(i,J)=Confusion(i,J)+1;
    end
end
```

```
Confusion=100*Confusion/Ntest;
AverageCorrect=sum( diag(Confusion) )/C

% Computing Classification Error (optional!)
% Error=zeros(C,C);
% for k=1:Ntrain
%     for i=1:C
%         for j=1:C
%             difference=Xtrain(:,k,i)-Myu(:,1,j);
%             f_xk_w(j)=exp(-0.5*difference' * SigmaInv(:,:,j) * difference) / const(j);
%         end
%         [m,J]=max(Pw.*f_xk_w);
%         Error(i,J)=Error(i,J)+1;
%     end
% end
% Error=(Error-diag(diag(Error)))/Ntrain;
% AveragePrError=sum(sum(Error))/C
```

MLPdfEstimator.m

```
[redacted]
   for k=1:Ntrain
      Sigma(:,:,i)=Sigma(:,:,i) + (Xtrain(:,k,i)-Myu(:,1,i)) * (Xtrain (:,k,i)-Myu(:,1,i))' ;
   end
end
Sigma=Sigma/(Ntrain-1);
```

6.2. Linear classifier based on L(L-1)/2 hyper planes

L_L_1LinearClassifier.m

```
%Linear classifier based on L(L-1)/2 hyperplanes----------------------------------------------
load Aircrafts\Data\Clean\Dubois_Glanz\Xtrain05
load Aircrafts\Data\Clean\Dubois_Glanz\Xtest05
%Xtrain(:,11:end,:)=[];

[L Ntrain C]=size(Xtrain);
Ntest=size(Xtest,2);
Pw(1:C)=1/C;        % A priori class distribution
Tolerance=0.01;     % Accuracy of W

k=1;
[redacted]
          zeros(C,(C+1));
Xtst=[ones(1,Ntest,C);Xtest];
for i=1:C
   g_x = W' * Xtst(:,:,i);
   [PosI PosK]=find(g_x >= 0);
   [NegI NegK]=find(g_x < 0);
   rowP=MapI(PosI);
   columnN=MapJ(NegI);
   VotesMat=zeros(C,Ntest);
   for k=1:size(PosI)
      VotesMat(rowP(k),PosK(k))=VotesMat(rowP(k),PosK(k))+1;
   end
   for k=1:size(NegI)
      VotesMat(columnN(k),NegK(k))=VotesMat(columnN(k),NegK(k))+1;
   end
   [MajorityVote,ClassNo]=max(VotesMat);
   ClassNo(find(MajorityVote==1))=C+1;
   for j=1:Ntest
```

```
      Confusion(i,ClassNo(j))=Confusion(i,ClassNo(j))+1;
   end
end
Confusion=100*Confusion/Ntest;
AverageUnKnown=sum(Confusion(:,C+1))/C
temp=Confusion;
temp(:,C+1)=[];
AverageCorrect=sum( diag(temp) )/C
```

LinearSeparator.m

```
function [W]=LinearSeparator(Class1,Class2,Tolerance)
[L N1]=size(Class1);
N2=size(Class2,2);
Z=([ones(1,N1) -ones(1,N2);Class1 -Class2])';
Myu1=sum(Class1,2)/N1;
Myu2=sum(Class1,2)/N2;
%Initial Values ------------
Bt=rand(N1+N2,1);
w0=0.5;
Wt=[w0;(Myu1-Myu2)];
Ro=0.01;
%---------------------------
error=100;
[redacted]
   =Wtt;
   Bt=Btt;
end
W=Wt;
```

6.3. Spectral classifier

- In this program lag selection is not employed in the test set.

SpectralClassifier.m

```
load Aircrafts\Data\Contaminated\NonRobustTrainSFArgs
load Aircrafts\Data\Contaminated\NonRobustTestSFArgs
load Aircrafts\Data\Contaminated\RobustLags
L=20;   % Limmiting frequency to Landa=2*pi*L/100
load Letters\Arial\Data\Clean\RobustTrainSFArgs
load Letters\BookAntiqua\Data\Clean\RobustTestSFArgs
load Letters\Arial\Data\Noisy\RobustLags

C=size(Phi_hat,3);   % Number of classes
N=size(Phi_hat,2);   % Number of test data in each class
[redacted]
      r=1:C
         f_tilda=SigmaZ2_tilda(r) * g(Phi_tilda(1:M(r),r), Landa, J(1:M(r),r));
         dw_tilda_hat=sum( h( f_tilda./f_hat ) ) / size(Landa,2);
         dw_hat_tilda=sum( h( f_hat./f_tilda ) ) / size(Landa,2);;
         Dw_bar(r)= dw_tilda_hat + dw_hat_tilda;
      end
      [u l]=min(Dw_bar);
      Confusion(k,l)=Confusion(k,l)+1;
   end
   k
end
AverageCorrect=mean(diag(Confusion))
```

- In this program lag selection is employed in the test set.

SpectralClassifierWithLS.m

```
load Aircrafts\Data\Contaminated\RobustTrainSFArgs
load Aircrafts\Data\Contaminated\RobustLags
load Aircrafts\Data\HubersParameter
load Aircrafts\Data\Contaminated\TestSeries
L=20;   % Limmiting frequency to Landa=2*pi*L/100
M_Init=30;  % Initial Model Order(for lag selection in test data)
Beta=0.2;   % Balance factor in Loss Function( // )

C=size(Yt,3);   % Number of classes
N=size(Yt,1);   % Number of test data in each class
Confusion=zeros(C,C);
g=inline('1./(2*pi*(abs(1-sum((Phi_tilda*ones(1,size(Landa,2))).*exp(-i*J*Landa)))).^2)','Phi_tilda','Landa','J');
h=inline('(x-1).^2','x');
[redacted]

      % Non-Robust ----------------------------------------------------
%       SigmaX_hat=sqrt(var(Y));
[redacted]

      [redacted]SigmaZ2_hat * g(Phi_hat, Landa, Lags);
      for r=1:C
         f_tilda=SigmaZ2_tilda(r) * g(Phi_tilda(1:M(r),r), Landa, J(1:M(r),r));
         dw_tilda_hat=sum( h( f_tilda./f_hat ) ) / size(Landa,2);
         dw_hat_tilda=sum( h( f_hat./f_tilda ) ) / size(Landa,2);;
         Dw_bar(r)= dw_tilda_hat + dw_hat_tilda;
      end
      [u l]=min(Dw_bar);
      Confusion(k,l)=Confusion(k,l)+1;
   end
   k
end
AverageCorrect=mean(diag(Confusion))
```

7. Evaluating the robust parameter estimation algorithms through spectral functions

- This program is used in section 4.5.1 to estimate the spectral functions shown in Figs. 14 and 15.
- It must be noted that here we use the lag structure obtained for noisy data (k=0) for contaminated data (k=12) too, as we want to examine only the performance of robust parameter estimation procedure (section 4.2)

SFTest.m

```
load Aircrafts\Data\Noisy\RobustTrainSFArgs
Phi_tilda1=Phi_tilda;
SigmaZ2_tilda1=SigmaZ2_tilda;
load Aircrafts\Data\Contaminated\TrainSeries
load Aircrafts\Data\Noisy\RobustLags
load Aircrafts\Data\HubersParameter

Phi_tilda2=[];
SigmaZ2_tilda2=[];
for j=1:size(Yt,3);
   Jj=J(1:M(j),j);
   for i=1:size(Yt,1)
      Y=Yt(i,1:T(i,j),j);

      % Non-Robust ----------------------------------------------
```

```
%       SigmaX_hat=sqrt(var(Y));
%       [Phi(:,i) SigmaZ2(i)]=CSARModel(Y/SigmaX_hat,Jj);
        [redacted]

    [redacted]
    SigmaZ2_tilda2(j)=mean(SigmaZ2);
    clear Phi
    j
end
% Once the parameters have been estimated all the lines above can be commented
% Spectral function estimation and show ------------------------------
g=inline('1/(2*pi*(abs(1-sum(Phi_tilda.*exp(-i*Landa*J))))^2)','Phi_tilda','Landa','J');
k=8;    % Number of class being tested: 1:Buccaneer 2:F16 ... 8:Starship
Jk=J(1:M(k),k);

Phik1=Phi_tilda1(1:M(k),k);
Sigk1=SigmaZ2_tilda1(k);
Phik2=Phi_tilda2(1:M(k),k);
Sigk2=SigmaZ2_tilda2(k);

Landa=[0:2*pi/100:2*pi*50/100];
L=[1:51];
for i=1:size(Landa,2)
    f1(i)=Sigk1*g(Phik1,Landa(i),Jk);
    f2(i)=Sigk2*g(Phik2,Landa(i),Jk);
end
plot(L,f1,'-',L,f2,':r');
xlabel('L (STARSHIP)');
ylabel('SPECTRAL FUNCTION');
```

CD Contents

- **Project**
  - **Aircrafts**
    - Buccaneer.bmp
    - F16.bmp
    - Goose.bmp
    - Harrier.bmp
    - Hawkeye.bmp
    - Hercules.bmp
    - Mirage.bmp
    - Starship.bmp
    - **Data**
      - HubersParameter
      - **Clean**
        - TestSeries
        - TrainSeries
        - SigmaX
        - **Dubois_Glanz**
          - Xtest02
          - Xtest03
          - Xtest04
          - Xtest05
          - Xtest06
          - Xtest07
          - Xtest08
          - Xtest09
          - Xtest10
          - Xtrain02
          - Xtrain03
          - Xtrain04
          - Xtrain05
          - Xtrain06
          - Xtrain07
          - Xtrain08
          - Xtrain09
          - Xtrain10
      - **Noisy**
        - NonRobustDGXtest
        - NonRobustDGXtrain
        - NonRobustTestSFArgs
        - NonRobustTrainSFArgs
        - RobustDGXtest
        - RobustDGXtrain
        - RobustLags
        - RobustTestSFArgs
        - RobustTrainSFArgs
        - TestSeries
        - TrainSeries
      - **Contaminated**
        - NonRobustDGXtest
        - NonRobustDGXtrain
        - NonRobustTestSFArgs
        - NonRobustTrainSFArgs
        - RobustDGXtest
        - RobustDGXtrain
        - RobustLags
        - RobustTestSFArgs
        - RobustTestSFArgs05
        - RobustTrainSFArgs
        - RobustTrainSFArgs05
        - TestSeries
        - TrainSeries

- **Letters**
  - **Arial**
    - G.bmp
    - H.bmp
    - I.bmp
    - J.bmp
    - **Data**
      - HubersParameter
      - **Clean**
        - RobustTrainSFArgs
        - TestSeries
        - TrainSeries
        - SigmaX
        - **Dubois_Glanz**
          - Xtest02
          - Xtest03
          - Xtest04
          - Xtest05
          - Xtest06
          - Xtest07
          - Xtest08
          - Xtest09
          - Xtest10
          - Xtrain02
          - Xtrain03
          - Xtrain04
          - Xtrain05
          - Xtrain06
          - Xtrain07
          - Xtrain08
          - Xtrain09
          - Xtrain10
      - **Noisy**
        - NonRobustDGXtest
        - NonRobustDGXtrain
        - NonRobustTestSFArgs
        - NonRobustTrainSFArgs
        - RobustDGXtest
        - RobustDGXtrain
        - RobustLags
        - RobustTestSFArgs
        - RobustTrainSFArgs
        - TestSeries
        - TrainSeries
      - **Contaminated**
        - NonRobustDGXtest
        - NonRobustDGXtrain
        - NonRobustTestSFArgs
        - NonRobustTrainSFArgs
        - RobustDGXtest
        - RobustDGXtrain
        - RobustLags
        - RobustTestSFArgs
        - RobustTrainSFArgs
        - TestSeries
        - TrainSeries
  - **BookAntiqua**
    - G.bmp
    - H.bmp
    - I.bmp
    - J.bmp
    - **Data**
      - **Clean**
        - RobustTestSFArgs
        - SigmaX
        - TestSeries
        - Xtest02
        - Xtest03
        - Xtest04
        - Xtest05
        - Xtest06
        - Xtest07
        - Xtest08
        - Xtest09
        - Xtest10
  - **Tahoma**
    - G.bmp
    - H.bmp
    - I.bmp
    - J.bmp

- **Data**
  - **Clean**
    - RobustTestSFArgs
    - SigmaX
    - TestSeries
    - Xtest02
    - Xtest03
    - Xtest04
    - Xtest05
    - Xtest06
    - Xtest07
    - Xtest08
    - Xtest09
    - Xtest10

- **TimesNewRoman**
  - G.bmp
  - H.bmp
  - I.bmp
  - J.bmp
  - **Data**
    - **Clean**
      - RobustTestSFArgs
      - SigmaX
      - TestSeries
      - Xtest02
      - Xtest03
      - Xtest04
      - Xtest05
      - Xtest06
      - Xtest07
      - Xtest08
      - Xtest09
      - Xtest10

- **MachineParts**
  - Bar.bmp
  - Bolt.bmp
  - Boomerang.bmp
  - Cylinder.bmp
  - Nut.bmp
  - Spanner.bmp
  - Star.bmp
  - **Data**
    - HubersParameter
    - **Clean**
      - TestSeries
      - TrainSeries
      - SigmaX
      - **Dubois_Glanz**
        - Xtest02
        - Xtest03
        - Xtest04
        - Xtest05
        - Xtest06
        - Xtest07
        - Xtest08
        - Xtest09
        - Xtest10
        - Xtrain02
        - Xtrain03
        - Xtrain04
        - Xtrain05
        - Xtrain06
        - Xtrain07
        - Xtrain08
        - Xtrain09
        - Xtrain10
    - **Noisy**
      - NonRobustDGXtest
      - NonRobustDGXtrain
      - NonRobustTestSFArgs
      - NonRobustTrainSFArgs
      - RobustDGXtest
      - RobustDGXtrain
      - RobustLags
      - RobustTestSFArgs

        - RobustTrainSFArgs
        - TestSeries
        - TrainSeries
    - **Contaminated**
        - NonRobustDGXtest
        - NonRobustDGXtrain
        - NonRobustTestSFArgs
        - NonRobustTrainSFArgs
        - RobustDGXtest
        - RobustDGXtrain
        - RobustLags
        - RobustTestSFArgs
        - RobustTestSFArgs05
        - RobustTrainSFArgs
        - RobustTrainSFArgs05
        - TestSeries
        - TrainSeries
- BayesClassifier.m
- BoundaryExtraction.m
- CFARModel.m
- CSARModel.m
- DataContaminator.m
- DGDataGenerator.m
- Imclean.m
- L_L_1LinearClassifier.m
- LagSelection.m
- LinearSeperator.m
- MLPdfEstimator.m
- RobustCSARModel.m
- RobustLags.m
- SFArgumentsEstimator.m
- SFArgumentsEstimator_Test.m
- SFTest.m
- SpectralClassifier.m
- SpectralClassifierWithLS.m
- TimeSeries.m
- TrainDataGeberator.m
- TSGenerator.m

❖ **References**

- [1].pdf
- [3].pdf

❖ Report.pdf