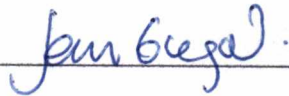


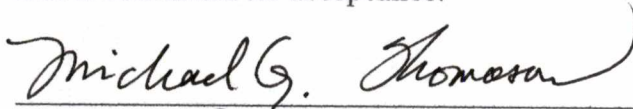
To the Graduate Council:

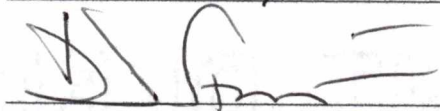
I am submitting herewith a thesis written by Joel Wesley Reed entitled "Markov Network Modeling and Recognition of Closed Contours." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.



Dr. Jens Gregor, Major Professor

We have read this thesis  
and recommend its acceptance:





Accepted for the Council:



Associate Vice Chancellor  
and Dean of the Graduate School

# **Markov Network Modeling and Recognition of Closed Contours**

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Joel Wesley Reed

December 1995

## **Acknowledgments**

Special thanks go to Dr. Jens Gregor whose guidance, assistance, and gentle reminders about deadlines and such made this work possible. Thanks also go to Dr. David Straight and Dr. Michael G. Thomason for serving on my thesis committee.

Appreciation goes to my parents for giving me the support and motivation necessary to pursue this degree. It would not have happened without them.

I would like to thank all of the people with whom I work at Oak Ridge National Laboratory for their willingness to answer questions and endure my sometimes unusual work schedule.

Finally, I dedicate this thesis to Glenneth. Without her love and ability to endure my ups and downs during the year or so I worked on this project it would never have been completed.

## **Abstract**

This thesis presents a method for pattern classification of two dimensional objects. The method is divided into three parts: preprocessing, inference, and classification. Preprocessing consists of the processing necessary to take a raw image containing an object and obtain a string encoding representing the object. Inference is the process of taking multiple related sample encodings and using them to train a constrained Markov network (CMN) using dynamic programming. Finally, classification is the process of aligning an unknown sample string encoding with multiple already trained CMNs so that the unknown string may be classified. In this alignment, the unknown string is considered cyclic, so a technique called channeling is incorporated to compute the alignment with less computational complexity than using a brute-force method of computing a full alignment for each possible offset.

Experimentation of the method was performed using images of aircraft. Using various parameter values, statistical information concerning the inferred CMNs entropy and disagreement cost are presented and interpreted. Also presented, for various parameters settings, are statistics and interpretation of classification performance and mean rotational error of the cyclic alignment algorithm.

# Contents

|          |                                              |           |
|----------|----------------------------------------------|-----------|
| <b>1</b> | <b>Introduction</b>                          | <b>1</b>  |
| 1.1      | Experimental Evaluation . . . . .            | 3         |
| 1.2      | Thesis Outline . . . . .                     | 6         |
| <b>2</b> | <b>Preprocessing</b>                         | <b>8</b>  |
| 2.1      | Contour Extraction . . . . .                 | 9         |
| 2.2      | Equilateral Polygon Approximation . . . . .  | 14        |
| 2.3      | String Encoding . . . . .                    | 17        |
| <b>3</b> | <b>Inference</b>                             | <b>22</b> |
| 3.1      | Inference . . . . .                          | 23        |
| 3.2      | Probability Functions . . . . .              | 27        |
| <b>4</b> | <b>Classification</b>                        | <b>31</b> |
| <b>5</b> | <b>Constrained Markov Network Statistics</b> | <b>36</b> |

|          |                                                             |           |
|----------|-------------------------------------------------------------|-----------|
| <b>6</b> | <b>Experimental Results</b>                                 | <b>44</b> |
| 6.1      | Classification Performance . . . . .                        | 45        |
| 6.1.1    | Dynamic Segments vs. Static Segments . . . . .              | 45        |
| 6.1.2    | Performance vs. Static Polygon Segment Length . . . . .     | 47        |
| 6.1.3    | Performance vs. Dynamic Polygon Segment Quantity . . .      | 50        |
| 6.1.4    | Performance vs. Angle Quantization Divisions . . . . .      | 51        |
| 6.1.5    | P Probability Functions vs. A Probability functions . . . . | 51        |
| 6.2      | Rotational Statistics . . . . .                             | 52        |
| <b>7</b> | <b>Conclusion</b>                                           | <b>55</b> |
|          | <b>Bibliography</b>                                         | <b>57</b> |
|          | <b>Appendices</b>                                           | <b>60</b> |
| <b>A</b> | <b>Single Engine Aircraft Sample Contours</b>               | <b>61</b> |
| <b>B</b> | <b>Twin Engine Aircraft Sample Contours</b>                 | <b>67</b> |
| <b>C</b> | <b>Fighter Aircraft Sample Contours</b>                     | <b>73</b> |
|          | <b>Vita</b>                                                 | <b>78</b> |

# List of Tables

|     |                                                                        |    |
|-----|------------------------------------------------------------------------|----|
| 2.1 | Contour statistics . . . . .                                           | 13 |
| 5.1 | Average CMN entropy . . . . .                                          | 38 |
| 5.2 | Average disagreement count . . . . .                                   | 41 |
| 6.1 | Confusion matrix: Static polygon segments, P functions . . . . .       | 46 |
| 6.2 | Confusion matrix: Dynamic polygon segments, P functions . . . . .      | 46 |
| 6.3 | Confusion matrix: Static polygon segments, A functions . . . . .       | 46 |
| 6.4 | Confusion matrix: Dynamic polygon segments, A functions . . . . .      | 46 |
| 6.5 | Single engine aircraft: Mean rotational error (in segments) . . . . .  | 53 |
| 6.6 | Twin engine aircraft: Mean rotational error (in segments) . . . . .    | 53 |
| 6.7 | Fighter engine aircraft: Mean rotational error (in segments) . . . . . | 54 |

# List of Figures

|     |                                                                                         |    |
|-----|-----------------------------------------------------------------------------------------|----|
| 1.1 | Inference and classification data-flow . . . . .                                        | 4  |
| 1.2 | Agusta 260 aircraft top silhouette . . . . .                                            | 5  |
| 2.1 | Point $f(x,y)$ and its 8 neighbors . . . . .                                            | 11 |
| 2.2 | Contour following example . . . . .                                                     | 12 |
| 2.3 | Agusta 260 aircraft contour . . . . .                                                   | 13 |
| 2.4 | Forward-backward search example . . . . .                                               | 17 |
| 2.5 | Calculation of the interior angle . . . . .                                             | 19 |
| 2.6 | Quantization into 4 divisions . . . . .                                                 | 20 |
| 2.7 | Agusta 260 aircraft EPA and reconstructed after 18 division quan-<br>tization . . . . . | 21 |
| 3.1 | Example CMN . . . . .                                                                   | 23 |
| 4.1 | Channeling property . . . . .                                                           | 33 |
| 4.2 | Recursive channeling alignment . . . . .                                                | 34 |

|     |                                                                      |    |
|-----|----------------------------------------------------------------------|----|
| 5.1 | Entropy: Dynamic polygon segments, P functions . . . . .             | 39 |
| 5.2 | Entropy: Dynamic polygon segments, A functions . . . . .             | 40 |
| 5.3 | Disagreement count: Dynamic polygon segments, P functions . . .      | 42 |
| 5.4 | Disagreement count: Dynamic polygon segments, A functions . .        | 43 |
| 6.1 | Correctly classified: Static polygon segments, P functions . . . . . | 48 |
| 6.2 | Correctly classified: Dynamic polygon segments, P functions . . .    | 48 |
| 6.3 | Correctly classified: Static polygon segments, A functions . . . . . | 49 |
| 6.4 | Correctly classified: Dynamic polygon segments, A functions . . .    | 49 |

# Chapter 1

## Introduction

Most pattern classification algorithms use either a statistical approach, a structural approach, or some combination. In statistical pattern recognition algorithms, characteristic measurements are extracted from the input data and are used to create a class-conditioned set of probabilities and/or probability density functions. The syntactic pattern recognition approach extracts structural information from the input data and assesses structural similarity of patterns. The most common structural approach is to relate the structure of patterns with the syntax of a formal language [9].

The objective of this thesis is to, in a new way, apply well known techniques and algorithms for contour extraction, polygon approximation based string encoding, dynamic programming inference and use of Markov networks for modeling and classification of two dimensional objects. The intent of this work is to create an

algorithm for pattern classification with reasonable recognition performance. In this context, pattern classification is defined as a method by which a computer, after having been trained with some samples from multiple classes of objects, can with some certainty decide that a never seen before object fits into one of the classes of objects.

The method presented here involves training multiple constrained Markov networks (CMN) with string encodings representing different classes of two dimensional objects. Each CMN is trained with a set of strings obtained from objects that have some common characteristic so that the CMN is encouraged to produce strings that have the common traits with high probability and other strings with lower probability. Later, a string of unknown class can be experimentally matched with each of these CMNs and is said to have the class of the CMN which can produce it with the highest probability.

This method is based on a three part training and classification process. The first part of the process is called *preprocessing* and includes all of the processing necessary to take a raw image of an object and turn it into a string encoding based on an equilateral polygon approximation of the object. A contour following algorithm is applied to the image of the object to obtain a set of points that define the contour of the object [7]. These points are used in the calculation of an equilateral polygon approximation (EPA) and then the list of interior angles of this EPA is quantized to form a string encoding of the object [8]. The second part

of the process, which is called *inference*, uses string encodings of related objects to iteratively train a constrained Markov network that can later be used to recognize string encodings with similar characteristics [2]. Finally, the third part of the process is called *classification* and is where a string encoding of an unknown object is experimentally matched against several already trained CMNs to see which can produce it with the highest probability, thereby making it possible to classify the object. The unknown string encoding is considered to be cyclic, so a technique called channeling is used to compute the alignment with less computational complexity than an exhaustive search [5]. Figure 1.1 illustrates the parts and data-flow within the training and classification process.

## 1.1 Experimental Evaluation

An important part of this work is experimental evaluation of this method. For the experimental evaluation, images of aircraft were used. The aircraft images were obtained from the 1991-1992 edition of “Jane’s All The World Aircraft” [4]. This book lists the aircraft with most of their specifications, gray-scale images, and top, front, and side silhouettes. Figure 1.2 is an example of a top silhouette like those used for experimentation.

A flat-bed scanner was used to scan the silhouettes straight from the book. Once the silhouettes were scanned, they were rotated so that the nose of the air-

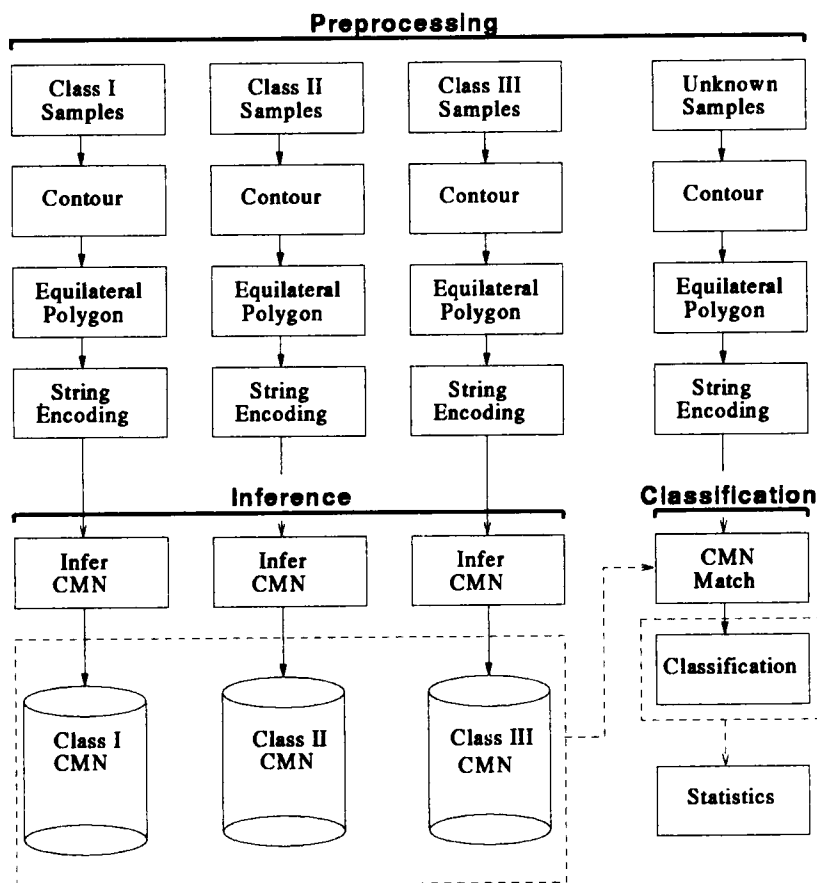


Figure 1.1: Inference and classification data-flow

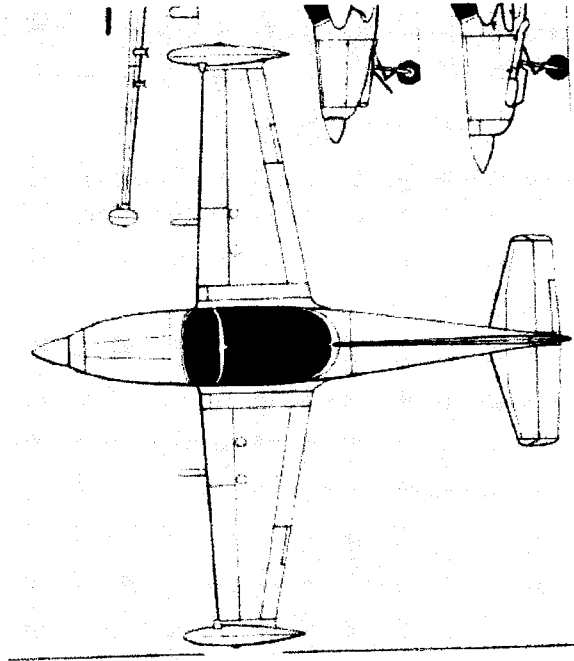


Figure 1.2: Agusta 260 aircraft top silhouette

craft pointed left (significance explained later) and then they were scaled relative to one another using the length and wingspan specifications. These samples were then divided into the following three classes:

1. Single engine aircraft: 19 samples (refer to appendix A)
2. Twin engine aircraft: 19 samples (refer to appendix B)
3. Fighter aircraft: 25 samples (refer to appendix C)

For each iteration of an experiment, a single sample was set aside as the sample for the test match, then a CMN was inferred for each class from the remaining strings, and then a match was done so that a classification for the test string could be obtained. This process continues until each sample has been used for a test

match. Several sets of experiments were run using various preprocessing parameters including equilateral polygon segment length and divisions of quantization on the interior angles of the polygon during string encoding. Also, experiments were run with two sets of probability functions which are used during the dynamic programming inference of the Markov networks. The results of these experiments along with some statistics gathered from the trained CMNs are presented later.

## 1.2 Thesis Outline

This thesis is organized into 7 chapters:

**Chapter 2** describes the parts of the preprocessing phase. A simple method for contour extraction from an image containing a single object, a method for obtaining an equilateral polygon approximation of a closed contour, and an algorithm for obtaining a string encoding from a polygon are all presented here.

**Chapter 3** describes the algorithm and associated probability functions that are used to infer a constrained Markov network from a set of strings.

**Chapter 4** describes an extension to the string-to-string method presented for matching a cyclical string to a pretrained constrained Markov network using a technique called *channeling*.

**Chapter 5** presents a set of statistics on the trained CMNs for various parameter settings in the preprocessing and two different sets of probability functions used during training.

**Chapter 6** presents results on classification performance for various preprocessing parameter settings and two different sets of probability functions used during training and matching. Also presented are statistics about rotational error in the CMN to cyclic string alignment.

**Chapter 7** concludes this thesis.

## Chapter 2

# Preprocessing

A one dimensional description of the sample objects is required by the classification method presented in this thesis, therefore the image of an object must be processed into a string encoding that captures the object's characteristics. This chapter describes the algorithms used in *preprocessing*, which include all of the processing necessary to take a raw image of a single object and turn it into a string encoding representing that object.

This process is divided naturally into three steps. In the first step, a contour extraction algorithm is applied to a raw image of an object. The output of the contour extraction algorithm is an ordered list of coordinates defining the outer closed contour of the object. In the second step, a method for equilateral polygon approximation (EPA) takes the list of coordinates and calculates a polygon with some desired polygon segment length. Finally, the EPA is used to create a string

encoding by calculating the ordered list of interior angles of adjacent pairs of polygon segments and then quantizing these to some number of angle divisions.

## 2.1 Contour Extraction

Processing an image containing a single object, in this case an aircraft, into an ordered list of points defining the outer shape of the object is a simple process, which takes place as follows:

1. Apply a global threshold function.
2. Locate an initial point.
3. Search for the remaining points in the contour.

A global threshold function is applied to the image because any image noise could cause problems during the contour extraction process. The following global threshold function is applied to every pixel  $(x,y)$  of the 256 gray-level image:

$$f(x,y) = \begin{cases} 0 & \text{if } f(x,y) < l \text{ (background)} \\ 255 & \text{if } f(x,y) \geq l \text{ (foreground)} \end{cases}$$

The threshold level  $l$  is set to a number that will suppress any noise in the image, while also setting all of the pixels making up the foreground object to 255. In the case of the aircraft images, applying the threshold function with  $l$  set to 127

removed all of the scanning noise and left a pure image of the aircraft in the foreground color.

This image is then used to find the contour points using the contour following algorithm presented in [7]. A contour is defined to be a sequence of points  $Y = y_0, \dots, y_{n-1}$ , where each point is an  $(x, y)$  pair representing a boundary point of the object and  $y_{i+1}$  is one of the 8-neighbors of  $y_i$ . When using the contour points for pattern classification, it is important that the initial point,  $y_0$ , of the contour start in a common location in all of the images. This is necessary so that common characteristics of the aircraft will be in approximately the same position in the resulting string encoding. With aircraft, the intuitive point to start the contour is at the nose. As mentioned above, all of the images have the nose of the aircraft pointing left, therefore, scanning for an initial point will start in the top left corner of the image and scan downward, moving to the right one column of pixels after the previous column is finished. Scanning continues until the first pixel is found that is set to the foreground gray-level. This will result in the initial point,  $y_0$ , of the contour being at the nose of the aircraft.

Once  $y_0$  has been located, the contour following algorithm can search for adjacent points in the sequence. Since the aircraft outline is a closed contour this process stops only when the initial point is found again during the search. Starting at initial point  $y_0$ , the algorithm maintains a current *direction* flag so that it can keep up with which direction the previous point in the sequence was found.

Figure 2.1 shows the pixel  $f(x, y)$  with its 8 neighbors and the direction numbers assigned to them.

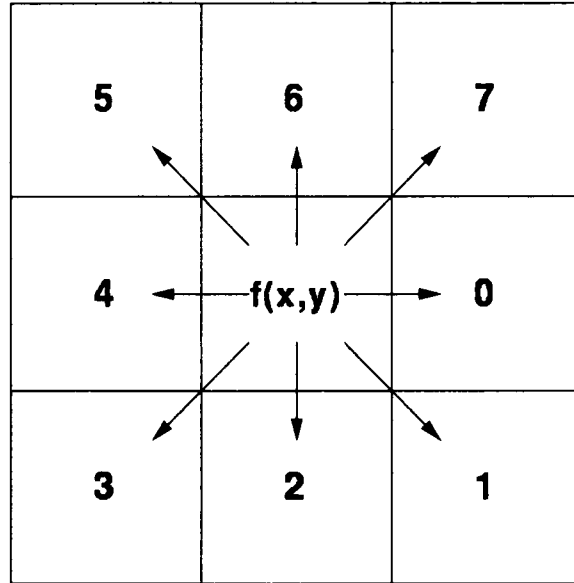


Figure 2.1: Point  $f(x,y)$  and its 8 neighbors

Since  $y_0$  is found in a left-to-right search the direction flag is set to 0 initially. Given a point,  $y_i$ , on the contour, it is impossible for the  $y_{i+1}$  contour point to be one of the 8-neighbors that was already checked in the the point search from  $y_i$ , which leaves 6 possible points. The first direction to search for the next contour point is always  $(direction+6)mod\ 8$  and then if not found there the search continues clockwise around the 8-neighborhood.

Figure 2.2 illustrates this process. Each foreground point of this portion of the image is symbolized by a filled circle. The dashed arrow shows where the search for the next contour point failed and the solid arrows show where the search was

successful. The numbers in the box show what the direction flag was set to at that pixel. So, for the initial pixel  $y_0$ , the direction flag is set to 0 which means that the search for the next contour point is going to be in the direction of  $(0 + 6) \bmod 8$  or 6 and proceed clockwise. Since the next contour point was found in the search of direction 7, the direction flag for that pixel is set to 7. The search proceeds from there with the initial direction of  $(7 + 6) \bmod 8$  which is 5.

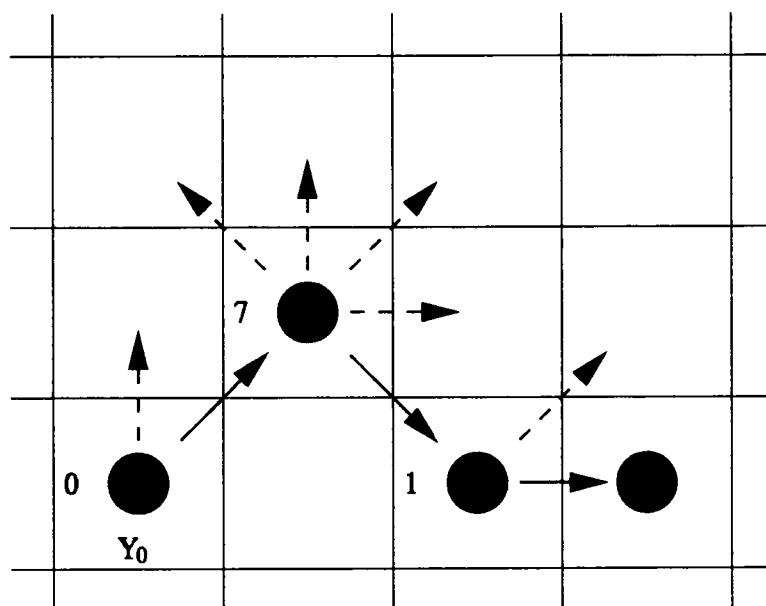


Figure 2.2: Contour following example

Figure 2.3 is an example contour derived from figure 1.2. Since all of the aircraft images are processed into contours, it is possible to gather some statistics concerning the result of this process for each aircraft class. Table 2.1 illustrates contour statistics for the three classes of sample data.

Table 2.1: Contour statistics

|         | Min. Points | Avg. Points | Max. Points |
|---------|-------------|-------------|-------------|
| Single  | 1816        | 2009        | 2504        |
| Twin    | 2038        | 2288        | 2758        |
| Fighter | 1492        | 2283        | 3358        |

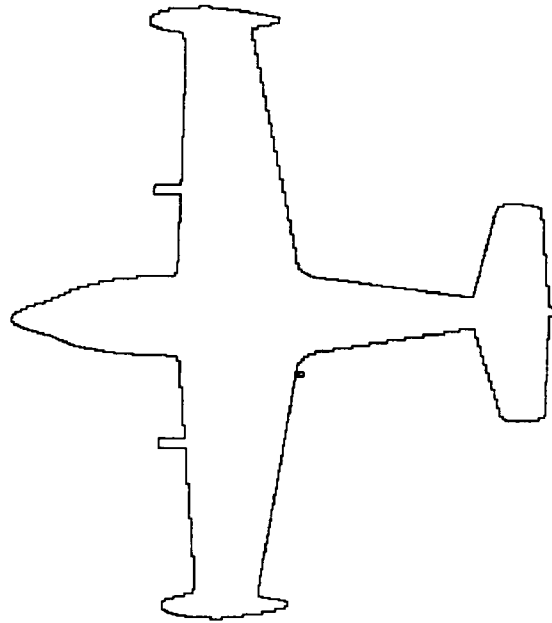


Figure 2.3: Agusta 260 aircraft contour

## 2.2 Equilateral Polygon Approximation

A method for equilateral polygon approximation (EPA) of closed contours that immediately lends itself to string encoding is presented by Rannou and Gregor in [8]. They describe a method to obtain an initial polygon that is very close to the optimal solution and then go on to describe another method to optimize this initial polygon. Their experimentation reveals that the method used to obtain the initial polygon results in a very good approximation of it, so this is what will be used for the equilateral polygon approximation here. Below, their method for obtaining this initial polygon is described.

To seed the EPA algorithm it is necessary to obtain a list of the dominate points in the contour, which are the points with high curvature as found at corners or bends. First, a copy of the contour point list is processed with Gaussian smoothing. The sequences of  $x$  and  $y$ -coordinates of the contour points are independently convolved with a symmetric one-dimensional filter of the form  $[a_w \cdots a_1 a_0 a_1 \cdots a_w]$  where

$$a_r = \frac{1}{\sqrt{2\pi}\sigma_s} \exp \left[ -\frac{1}{2} \left( \frac{r}{\sigma_s} \right)^2 \right]$$

The width  $w$  determines the accuracy of the filter and the standard deviation  $\sigma_s$  controls the level of smoothness. Note that the Gaussian smoothed contour points have a one-to-one correspondence with the contour points.

Next, the curvature is calculated at each point of the Gaussian smoothed con-

tour using a curvature approximation method with a threshold to avoid near-zero curvature maxima. The mean and variance are then calculated for the list of curvatures. Any point from the original contour point list that has a curvature greater than the mean plus twice the variance is saved and all of the others are discarded. Note that the desired length of the equilateral polygon approximation has not been used anywhere in the previous calculations. Any of the points obtained that are within approximately 2 times the desired polygon segment length of another point are discarded also. This results in a polygon approximation of the contour with vertices on the dominant points, all vertex points at least twice the desired segment length apart, and possibly areas where adjacent vertexes are significantly more than twice the desired segment length apart.

More vertices must be filled in to obtain the approximately equilateral polygon. Each pair of adjacent dominant points define a contour segment. An iterative forward-backward search is carried out in each contour segment to fill in more vertices. Let  $s_i$  and  $s_j$  be a pair of adjacent dominant points with  $c_i$  and  $c_j$  as their corresponding contour points. Furthermore, let  $s_f$  and  $s_b$  denote respectively the last vertices found when searching forward and backward; initially  $s_f = s_i$  and  $s_b = s_j$ . The forward-backward search, as taken from [8], then takes place as follows:

- Scan forward along the contour going from  $s_f$  toward  $s_b$  until finding a  $c_k$  for which  $\|s_f - c_{k-1}\| \leq L < \|s_f - c_k\|$  where  $L$  is the desired polygon segment

length. A new polygon vertex  $s_r$  can be located at  $c_{k-1}$  or  $c_k$  and, in either case, result in an edge which is almost the desired length.

- Let  $\xi = L - \|s_f - c_{k-1}\|$  and  $\zeta = \|s_f - c_k\| - L$  denote the negative and positive edge length errors and let  $E_f$  be the accumulated edge length error.
- If  $|E_f - \xi| < |E_f + \zeta|$  then set  $E_f = E_f - \xi$  and assign the new vertex  $s_r$  to  $c_{k-1}$ , otherwise set  $E_f = E_f + \zeta$  and assign  $s_r$  to  $c_k$ . The idea is to allow the new edge to be long only if the previous one was short, and vice versa so that  $E_f$  is kept small.
- Scan backward along the contour going from  $s_b$  toward  $s_f$  until finding a  $c_k$  for which  $\|s_b - c_k\| \leq L < \|s_b - c_{k-1}\|$ . A separate counter  $E_b$  is maintained for the accumulated edge length error. Assign new vertex  $s_r$  to  $c_{k-1}$  as described above and update  $s_b$  accordingly.

This forward-backward search is repeated until a  $c_k$  can no longer be found at which point we check whether  $s_f$  and  $s_b$  lie too close together. If  $\|s_f - s_b\| \leq 0.8 \times L$ , then  $s_f$  and  $s_b$  are replaced by a single vertex which is equidistant to both of them. If not, then the vertex configuration is accepted and the next contour segment is considered. When all contour segments have been processed, the polygon initialization terminates. Figure 2.4(a) illustrates a forward iteration of the EPA algorithm, figure 2.4(b) shows a backward iteration, and figure 2.4(c) shows a terminating iteration. Also see the figure on page 21, which is the result

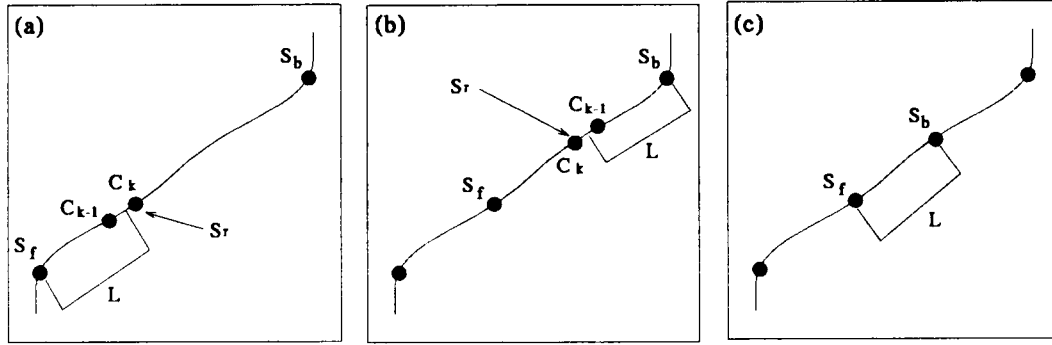


Figure 2.4: Forward-backward search example

of this algorithm when applied to an aircraft contour with  $L$  set to 25.

## 2.3 String Encoding

Obtaining a string encoding from the polygon approximation is achieved by creating an ordered list of interior angles between adjacent pairs of polygon segments and then quantizing each element of this list by some number of angle divisions. This results in the one-dimensional representation of the object that is necessary for pattern classification. Quantization of the angle list results in rounding of the angle measures, possibly a smaller alphabet for the string encoding, removes data noise, and improves the performance of the pattern classification algorithm which is presented later.

Let  $s_j$ ,  $s_k$ , and  $s_l$  be adjacent vertices in the equilateral polygon approximation and  $A_k$  is the angle between  $\overline{s_k s_j}$  and  $\overline{s_k s_l}$  that lies on the side interior to the polygon. It is possible to use the following equation for the inner product of two

vectors to find the smaller angle between them:

$$\langle \overline{s_k s_j} \rangle \cdot \langle \overline{s_k s_l} \rangle = \|\overline{s_k s_j}\| \|\overline{s_k s_l}\| \cos \theta$$

Even so, some other technique must be used to decide whether the interior angle  $A_k = \theta$  or  $A_k = 2\pi - \theta$ . Therefore, the following method is used to find the interior angle  $A_k$ :

- Translate all three of the points so that  $s_j$  is at the origin;  $s_j = (0,0)$ ,  $s_k = s_k - s_j$ , and  $s_l = s_l - s_j$ .
- Calculate the angle,  $\theta$ , between  $\overline{s_j s_k}$  and the X-axis. See figure 2.5(a).
- Perform a two-dimensional rotation of  $-\theta$  on  $s_k$  and  $s_l$  to bring  $s_k$  onto the X-axis.
- Calculate the angle,  $\vartheta$ , between  $\overline{s_k s_l}$  and the X-axis. See figure 2.5(b).
- Given that the contour runs in a counter-clockwise direction, the interior angle,  $A_k = \pi - \vartheta$  if  $s_l$  is above, or on, the X-axis. Otherwise, if  $s_l$  is below the X-axis,  $A_k = \pi + |\vartheta|$ .

Repeat this for each vertex in the equilateral polygon approximation to obtain an ordered list of interior angles.

Next, the list of interior angles must be quantized to round them to integers and at the same time possibly obtain a smaller alphabet for the string encoding.

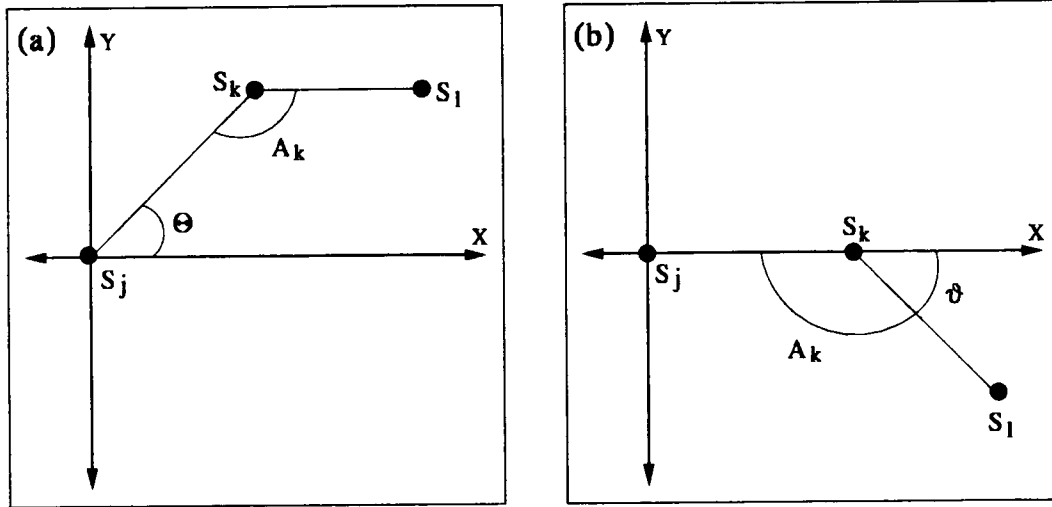


Figure 2.5: Calculation of the interior angle

Let  $d$  be the desired number of quantization divisions, then  $\alpha = 2\pi/d$  is the radian measure of each division. Then, given an angle  $A$ , the quantized angle  $A' = \lceil ((A + (\alpha/2))/\alpha) \rceil$ . Applying this equation to each angle in the angle list will result in a list of quantized angles such that each is between and including 1 and  $d$ . Figure 2.6 illustrates quantization into 4 divisions. Figure 2.7 shows an aircraft before quantization and shows the same aircraft after being quantized using 18 divisions and then reconstructed.

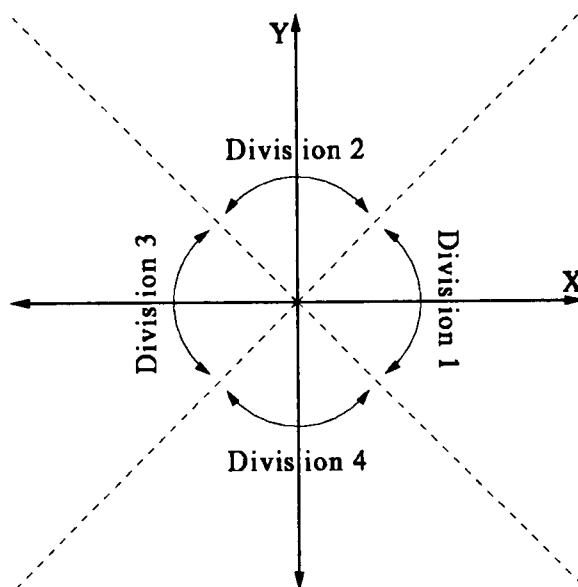


Figure 2.6: Quantization into 4 divisions

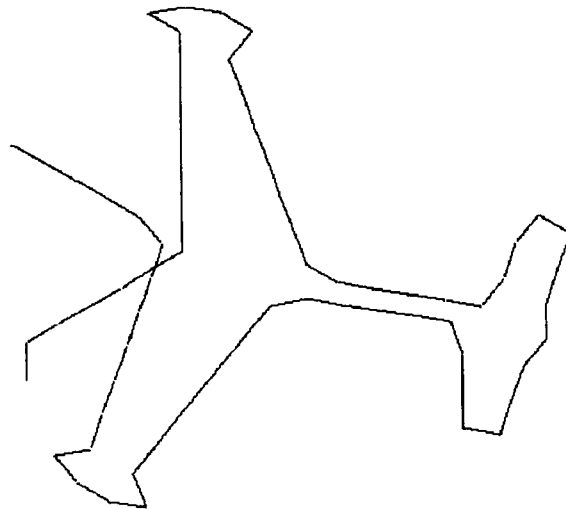
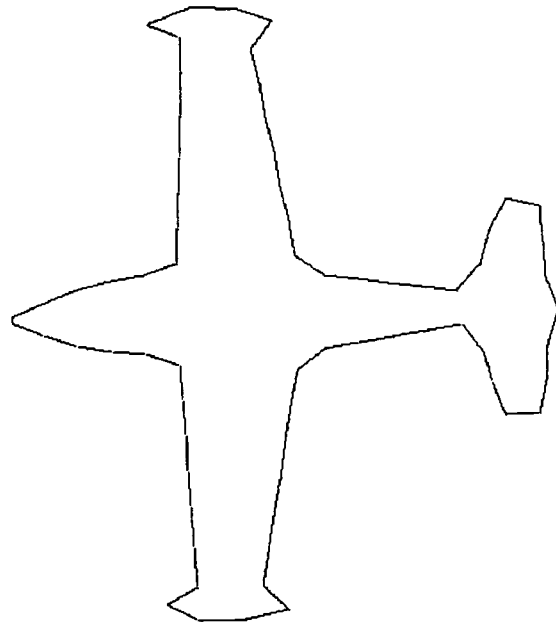


Figure 2.7: Agusta 260 aircraft EPA and reconstructed after 18 division quantization

## Chapter 3

# Inference

The topology of a constrained Markov network (CMN) is fixed to be a concatenation of stages. In the context of this thesis, the stages contain a set of states, each of which represent either a symbol from the string alphabet  $\Sigma$ , or the bypass symbol  $\epsilon$ . Figure 3.1 illustrates the topology of a CMN.

In [2], Gregor, Thomason, and Guthrie present a method for inference of constrained Markov networks using dynamic programming for use in pattern analysis and pattern recognition. Dynamic programming is a technique whereby a large problem can be solved by combining the solution of a number of smaller problems. This technique is more efficient than an exhaustive search when used to compute the optimal alignment between a sample string and a CMN [10]. Computing the optimal alignment calculates how to optimally modify the CMN so that it can produce the sample string with the highest probability.

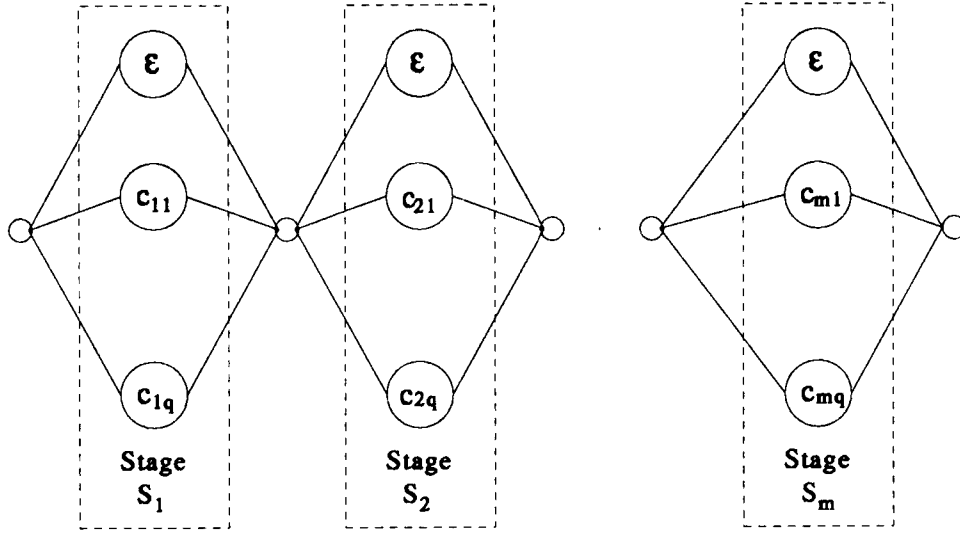


Figure 3.1: Example CMN

Computationally and in this discussion a CMN is a matrix where the rows are stages of the CMN and the columns are an ordered list of the symbols in the alphabet and the bypass symbol  $\epsilon$ . The ordering of the alphabet symbols allow the CMN and the matrix to be used interchangeably in this discussion. In the CMN matrix, each cell contains a frequency count, thus the probability of entering a particular state is the frequency count of that state divided by the sum of the contents of the row where that state resides. The regular structure of a CMN makes it computationally relatively inexpensive to use for pattern recognition [3].

### 3.1 Inference

Inferring a CMN from a finite list of strings consists of using a single sample string to create an initial CMN and then sequentially aligning, using DP, the remaining

strings to the CMN such that some criterion function is minimized. For each training string, first, the dynamic programming matrix for the alignment of the string and the CMN is calculated, and second, the CMN is modified according to an optimal alignment, of which there may be many.

A single string,  $T = t_1 t_2 \dots t_m$  from the set of training strings, which have a given alphabet of  $\Sigma = \{\epsilon, a_1, a_2, \dots, a_q\}$ , is used to create an initial CMN. For simplification of the matrix notation, the ordered alphabet  $O$  is made up of integers with  $\epsilon = 0$  and  $a_1 = 1, a_2 = 2, \dots, a_q = q$ . The initial CMN  $C$  is a  $m \times (q + 1)$  matrix where:

$$\forall i \in \{1, \dots, m\} \text{ and } \forall j \in \{0, 1, \dots, q\} \quad C_{ij} = \begin{cases} 0 & \text{where } j \neq T_i \text{ or } j = 0 \\ 1 & \text{where } j = T_i \end{cases}$$

The initial CMN  $C$  can produce string  $T$  with a probability of 1 and will produce all other strings with a probability of 0.  $F$  is the quantity of strings aligned with the CMN, which is equal to the sum of any row of  $C$ , and in the initial CMN  $F = 1$ . For example, if the training string  $T = 1231$  and the ordered alphabet  $O = \langle 0, 1, 2, 3 \rangle$  then the matrix representing the CMN is

$$C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

Once the initial CMN is obtained, the process of iteratively aligning other training strings with the CMN using dynamic programming commences. First, the DP matrix must be calculated and then information from the alignment can be used to modify the CMN via a sequence of edit operations which are defined by an optimal path in the DP matrix. There are three CMN edit operations: *insert* which corresponds to a horizontal move along the optimal path, *match* which corresponds to a diagonal move along the optimal path, and *bypass* which corresponds to a vertical move along the optimal path. An insert operation adds another stage to the CMN, a match operation increments the frequency count of the symbol matched, and a bypass operation increments the frequency count of the  $\epsilon$  state.

Given another training string  $T = t_1 t_2 \dots t_n$  and the initial CMN or some training iteration of the CMN,  $I(i, j)$ ,  $M(i, j)$ ,  $B(i)$  are functions that define the probability for insert, match, and bypass operations at state  $j$  in stage  $i$  of the CMN, respectively. For simplicity these functions will be called probability functions

hereafter and will be discussed in detail later. The size of the DP matrix  $D$  is  $m \times n$ ;  $m$  from the number of rows in the CMN matrix and  $n$  from the length of the training string. The DP matrix is initialized as follows:

- $D_{0,0} = 0$
- $D_{i,0} = 0$  for  $0 \leq i \leq m$
- $D_{0,j} = 0$  for  $0 \leq j \leq n$

The remaining entries are filled in with the following recursive function:

$$D_{ij} = \text{MAX} \begin{cases} D_{i,j-1} + \log(I(i,j)) & \text{Insertion (Horizontal)} \\ D_{i-1,j-1} + \log(M(i,j)) & \text{Match (Diagonal)} \\ D_{i-1,j} + \log(B(i)) & \text{Bypass (Vertical)} \end{cases}$$

Note that the  $\log$  function is used to prevent underflow.

Using the information from the DP matrix, the CMN is modified to accommodate the new string  $T$ . Let the optimal path in the DP matrix from  $(0,0)$  to  $(m,n)$  be indexed by  $(i,j)$  such that  $i$  denotes stage  $i$  in  $C$  which is equivalent to row  $i$  in the DP matrix and let  $j$  denote  $t_j$ , which is  $j^{\text{th}}$  symbol in  $T$ . It is important to note that since the alphabet has been given an order,  $j$  can be used as an index inside a stage of CMN  $C$ . Each step along the optimal path represents either an insertion, match, or bypass operation. The method uses this information to modify the CMN as follows:

**Insertion:** A new stage is inserted in  $C$  at  $i$  with all frequency counts initialized to 0 except for  $C_{ij} = 1$  and  $C_{i0} = F$  where  $F$  is the number of strings previously aligned with  $C$ . This creates a stage in the CMN where symbol  $j$  has a frequency count of 1 and all of the previously aligned strings are made to take the bypass state  $\epsilon$ .

**Match:**  $C_{ij}$  is incremented by 1. This results in making symbol  $j$  at stage  $i$  more likely.

**Bypass:**  $C_{i0}$  is incremented by 1. This makes the bypass of this state in stage  $i$  more likely.

Once the modification is complete, the resulting CMN  $C$  can be trained with more strings or be used for matching.

### 3.2 Probability Functions

Two different sets of probability functions were used during experimentation. As above, let  $C_{ij}$  be a CMN, and  $F$  be the number of strings already aligned,  $i$  is the current stage in the CMN, and  $j$  indexes the symbols in that stage.

The first set of  $I(i, j)$ ,  $M(i, j)$ , and  $B(i)$ , which take into account only fre-

quency count, are as follows:

$$I(i, j) = \frac{1}{F+1} \quad \text{Insertion (Horizontal)}$$

$$M(i, j) = \frac{C_{ij}+1}{F+1} \quad \text{Match (Diagonal)}$$

$$B(i) = \frac{C_{i0}+1}{F+1} \quad \text{Bypass (Vertical)}$$

Each of these functions can be explained intuitively by comparing them to their corresponding CMN modification. For insertion, a new stage is inserted into the CMN where a single symbol's frequency count is set to 1 and the  $\epsilon$  frequency count is set to  $F$ , therefore the probability of entering this single symbol's state would be  $\frac{1}{F+1}$ . The modifications to the CMN for matching in stage  $i$  with symbol  $j$  is to increment  $C_{ij}$ , thus the probability of entering this stage would be  $\frac{C_{ij}+1}{F+1}$ . Finally, for the modification to the CMN for a bypass is to increment  $C_{i0}$  which represents the  $\epsilon$  symbol in stage  $i$ . The probability of entering this state would then be  $\frac{C_{i0}+1}{F+1}$ . These probability functions look at the current CMN and calculate a probability as if the CMN modification had already been made.

Realizing that the strings of symbols being aligned are angle measures leads up to the second set of probability functions. If all of the symbols in the alphabet  $\Sigma \setminus \{\epsilon\}$  are fitted, in order, around a circle, then the symbols in the alphabet that lie closer to each other on the circle are also closer to the same angle. Using this information, the cost function for *Match* can be modified to give a lower probability if the symbol experimentally being matched is farther from the most

likely symbol in the current stage of the CMN. Note that no changes are made to the function for insertion because it is calculating the probability in a new stage where the most likely symbol will be the symbol that caused the insertion. Also no changes are made to the bypass function because it is calculating the probability of ignoring all of the symbols in a particular stage.

Given CMN  $C_{ij}$ , let  $J_i = \operatorname{argmax}_{\delta: \delta \neq 0} C_{i\delta}$  and  $t_j$  symbol under consideration from the training string  $T$ . As above  $q + 1$  is the size of the alphabet  $\Sigma$ . The distance  $d$  between  $J_i$  and  $t_j$  is defined as

$$d = \min\{|J_i - t_j|, |J_i + q - t_j|\}$$

If the symbols are the same then  $d = 0$  and if the symbols are exactly opposite to each other on the circle mentioned above, then  $d = \frac{q}{2}$ . In desiring to decrease the probability of a match when the symbols are far from each other, the following function is used for match:

$$M(i, j) = \frac{C_{ij} + 1}{F + 1} \times \left[1 - \frac{d}{q}\right] \quad \text{Match (Diagonal)}$$

If the symbols lie on top of each other then the original match probability is multiplied by  $1 - \frac{q}{q}$  which is 1 and if the symbols are opposite each other the original match probability is multiplied  $1 - \frac{q/2}{q}$  which is 0.5. As will be illustrated later in the section on experimental results, this set of probability functions performed better than the original set.

## Chapter 4

# Classification

The purpose of aligning an unknown sample string with a pretrained CMN is to obtain the path through the CMN that would generate the string with the highest probability. This probability corresponds to the likeliness that the string is of the same class as the CMN.

When trying to classify an unknown sample it is not desirable, or sometimes possible, to orient the sample to match the orientation of the training samples. This problem occurs if the EPA is encoded as a linear string, but is not a problem if the EPA is encoded as a cyclic string. Cyclic string encoding not only overcomes the orientation problem, but provides a tool to determine the orientation of the unknown object [6]. Matching a sample string with a pretrained CMN using dynamic programming is closely related to the CMN inference discussed earlier and the probability functions used for cyclical string matching are identical to the

ones used for inference.

Optimal alignment of a CMN of length  $m$ , and a linear string of length  $n$ , is computed in time  $O(mn)$  using DP, but if the string is cyclic, computational complexity increases accordingly [1]. A brute force method that considers all possible offsets in the cyclic string will compute the optimal alignment and offset in  $O(mn^2)$ . The optimal alignment and offset can be computed more efficiently than this. In [5], Maes presents an algorithm to optimally align a string with a cyclic string using dynamic programming with a computational complexity of  $O(mn \log n)$ . The algorithm uses a technique called channeling to reduce the computational complexity of each alignment. This technique is readily extensible to align a CMN and a string.

The channeling technique is able to reduce computational complexity by reducing the number of cells in the DP matrix that must be computed. Let  $C$  be an already trained CMN and let  $T_{1:n1:n}$  be a sample string that has been concatenated with itself. Further, let  $O_l$  define an optimal path in the DP matrix resulting from an alignment between CMN  $C$  and string  $T$  at offset  $l$  in  $T$  and let  $O_r$  define an optimal path resulting from an alignment between  $C$  and string  $T$  at offset  $r$  in  $T$ . Let  $O_x$  be an optimal path between  $C$  and  $T$  at offset  $x$  in  $T$  that has yet to be computed. The channeling property states if that the offsets in  $T$  have the property  $l < x < r$ , then an optimal alignment between CMN  $C$  and string  $T$  at offset  $x$  can be computed using only the matrix cells that lie inside the channel

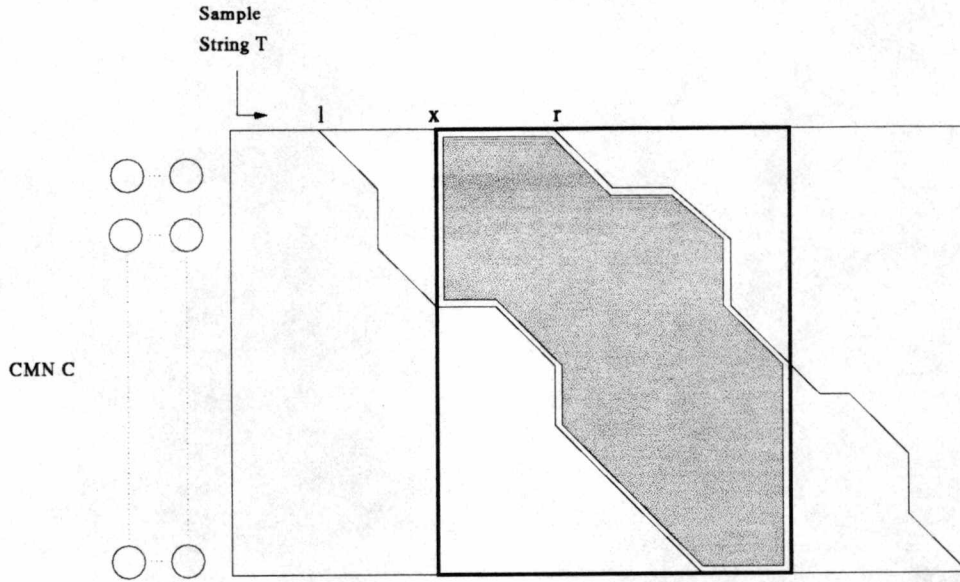


Figure 4.1: Channeling property

defined by  $O_l$  and  $O_r$ . Figure 4.1 shows the  $m \times 2n$  matrix resulting from the cyclical alignment of  $C$  with  $T$ . Notice, that to obtain  $O_x$  it is only necessary to compute matrix cells in the gray area of the figure.

Using the channeling property in the DP alignment of two strings,  $S$  and  $T$  is a recursive procedure. As before, let  $T_{1:n;1:n}$  be a sample string that has been concatenated with itself. Let string  $S$  have a length of  $m$ . The algorithm would initially calculate an alignment between the two strings at an offset of 0 in  $T$  resulting in an optimal path from  $(0,0)$  to  $(m,n)$  in the DP matrix. Since  $T$  is concatenated with itself, the optimal path from  $(0,n)$  to  $(m,2n)$  is identical to the path from  $(0,0)$  to  $(m,n)$ . The channeling property would be applied to find an optimal path at an offset halfway between 0 and  $n$  (figure 4.2(a)). Note, as above, the computation is necessary only in the DP matrix cells defined by the

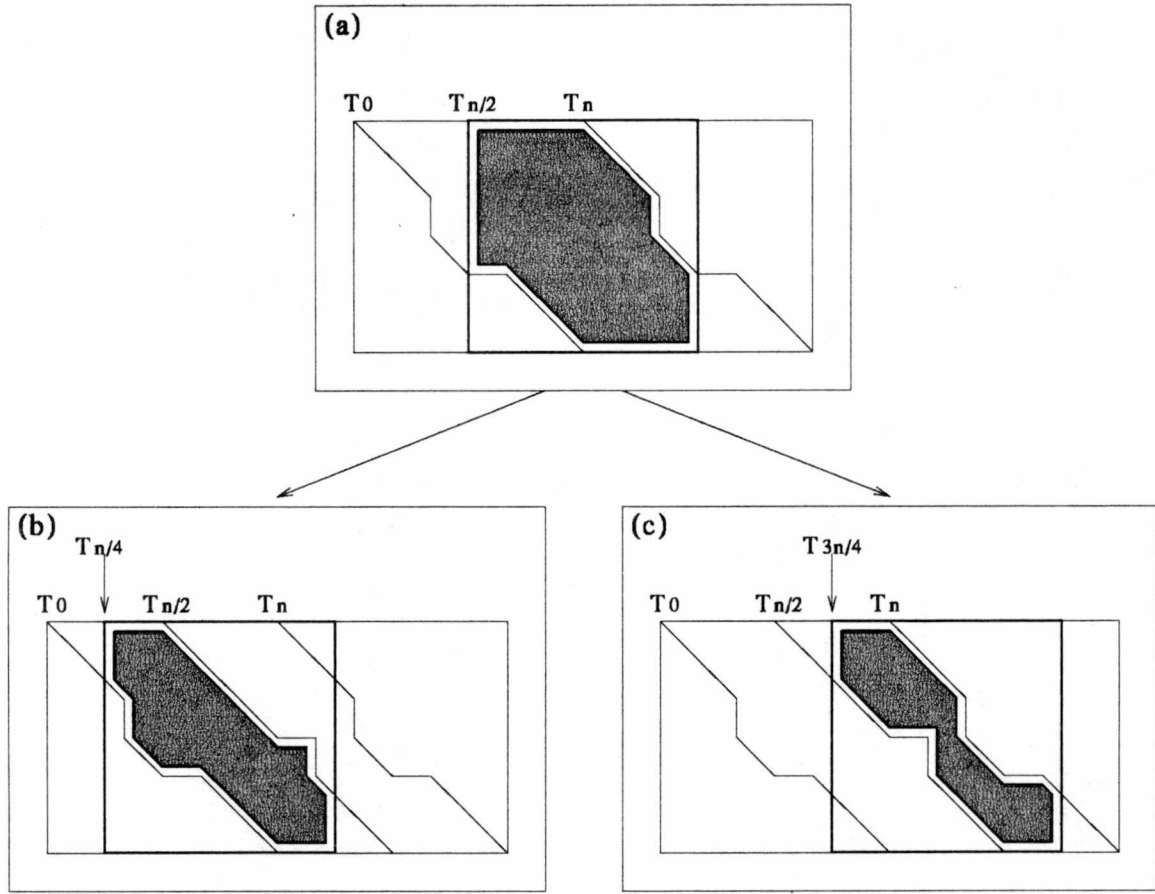


Figure 4.2: Recursive channeling alignment

gray areas in the figure. The  $\frac{n}{2}$  path allows for the channeling property to be applied to find the optimal path at two more offsets:  $\frac{n}{4}$  and  $\frac{3n}{4}$  (figure 4.2(b,c)). The algorithm would continue execution, while maintaining the offset of the most optimal alignment seen so far, until all possible offsets are exhausted.

Maes, in [5], gives a formal proof of the channeling property for string-to-cyclic-string alignment. In the alignment of a sample string with a CMN, the CMN is treated as if it were a string which means that since the channeling property is

proven for string-to-string alignment, it is valid for CMN-to-string alignment.

Given a unknown sample string, classification is accomplished by applying the process described above to align the string with multiple CMNs. The string is then said to have the same class as the training strings used to infer the CMN which resulted in the highest probability during alignment.

## Chapter 5

# Constrained Markov Network Statistics

In order to help interpret the classification results of this method it is important to look at some statistics of the inferred CMNs. For these statistics, the single engine CMNs were inferred using *all* of the single engine aircraft samples and likewise for the twin engine and fighter aircraft CMNs. Statistics were gathered for various parameter values. The modifiable parameters are:

1. The segments of the equilateral polygon approximation can specified in two ways. Either segments of a desired length, hereafter denoted *static segments*, or a desired number of segments, hereafter denoted *dynamic segments*.

2. If static segments are used in the EPA, then the segment length can be varied.

If dynamic segments are used in the EPA, then the number of segments per polygon can be varied.

3. The number of divisions in the angle quantization can be varied.

4. The probability functions used in inferring the CMN and in classification of sample strings can be simply probabilistic or take into account that the strings are a list of angles. Hereafter, probabilistic only functions will be denoted by *P functions* and functions that take into account angle distance will be denoted by *A functions*.

Two different CMN statistics, for each of the possible parameters, are presented in this chapter: entropy and disagreement count. Entropy measures the uncertainty of an entire probability distribution, which, when used to measure a CMN, might give a good indication of how well the CMN will perform as a classifier. Disagreement count is an error measure in quadratic form [2].

First, CMN entropy for various preprocessing parameters and inference probability functions is considered. Given that  $C$  is a CMN with  $m$  stages and  $q + 1$  (an alphabet of  $q$  plus one bypass state  $\epsilon$  which is stored in  $C_{m0} \forall m$ ) states within each stage where  $C_{ij}$  is the frequency count of symbol  $j$  in stage  $i$  and  $F$  is the

total quantity of strings aligned to infer  $C$ , then the entropy is defined as:

$$H(C) = - \sum_{i=1}^m \sum_{j=0}^q \frac{C_{ij}}{F} \log \frac{C_{ij}}{F}$$

Figure 5.1 represents entropy measurements, for each class, using the P functions for inference and figure 5.2 represents the same, but using the A functions for inference. Table 5.1 shows the entropy measurement averaged for all of the experimental parameter values. As the table illustrates, on average, the entropy

Table 5.1: Average CMN entropy

|         | P functions | A functions |
|---------|-------------|-------------|
| Single  | 77.2        | 74.8        |
| Twin    | 78.7        | 74.2        |
| Fighter | 85.9        | 83.4        |

for the set of A functions is slightly lower, which predicts that matching using them will yield better results. It is also important to note that the entropy of the CMNs inferred from the fighter aircraft samples are significantly higher than the other classes. This is because the fighter aircraft have much more diversity in their shape and contour length than the other aircraft. The statistics obtained using static segments are excluded here because they report similar results.

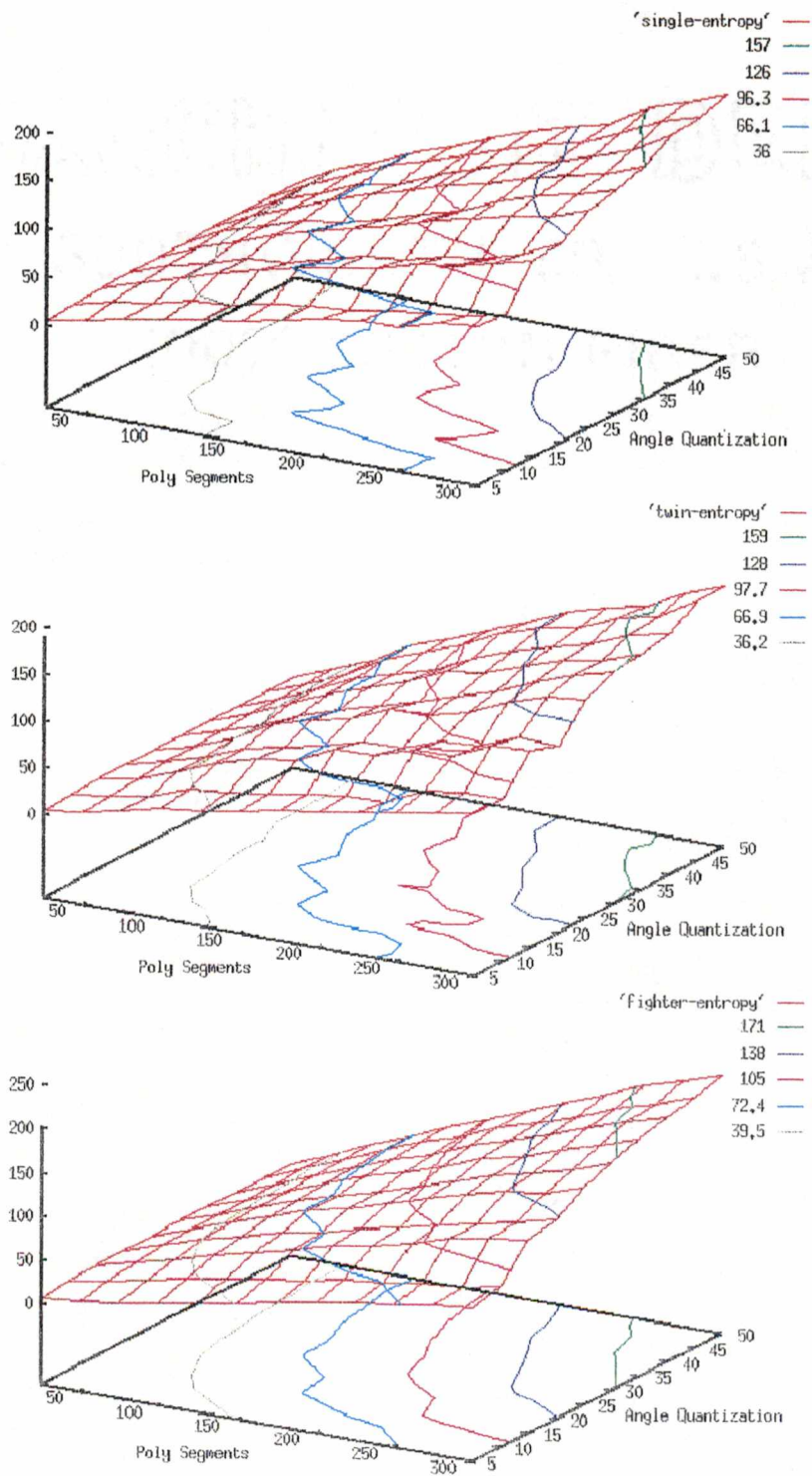


Figure 5.1: Entropy: Dynamic polygon segments, P functions

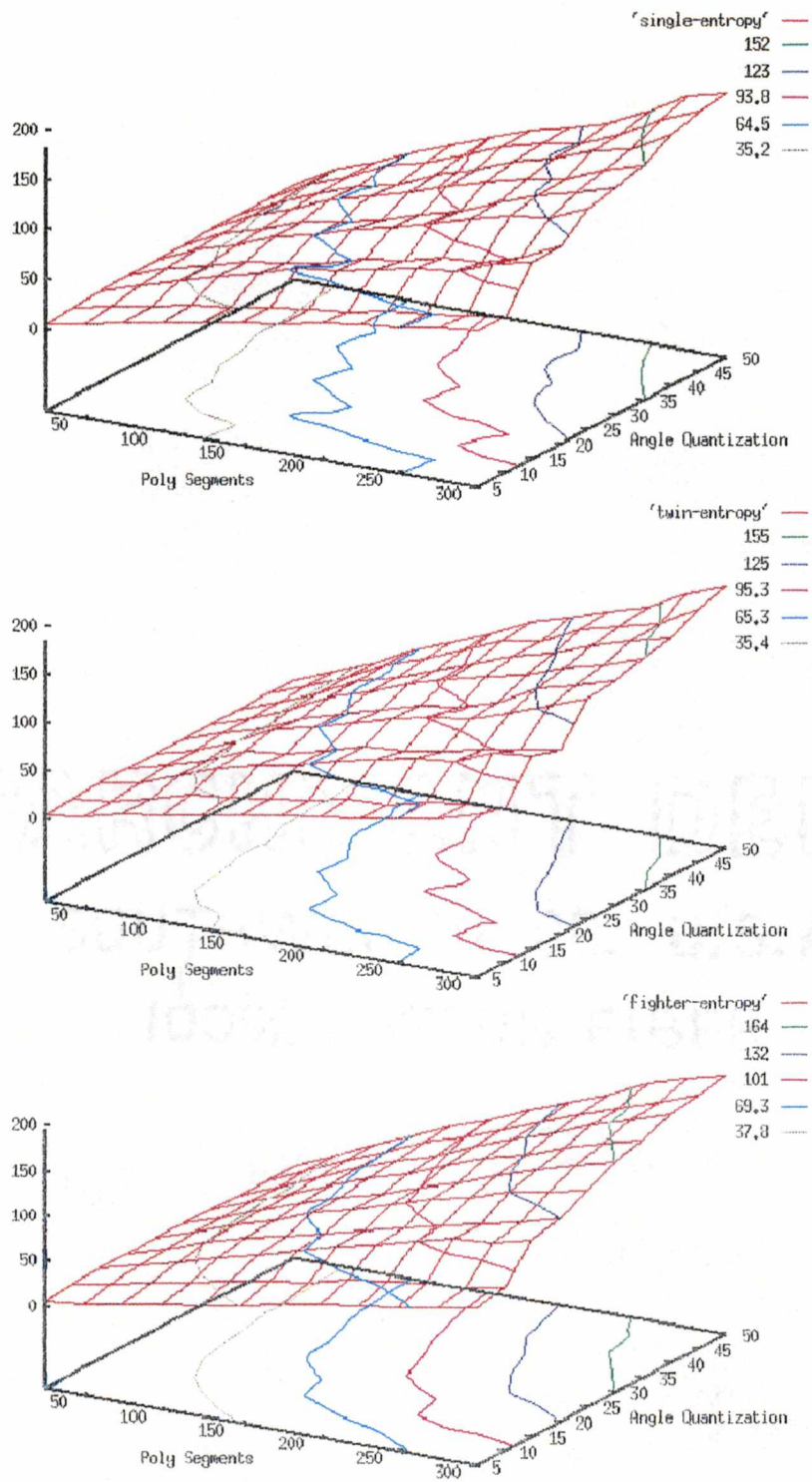


Figure 5.2: Entropy: Dynamic polygon segments, A functions

The disagreement count of CMN  $C$  is defined as

$$DC(C) = NF^2 - \sum_{i=1}^N \sum_{j=0}^q C_{ij}^2$$

where  $N$  is the number of stages in  $C$ ,  $q$  the number of states in each stage, and  $F$  is the number of strings that have been aligned with CMN  $C$  [2].

Figure 5.3 illustrates the disagreement cost of CMNs inferred with the P functions and figure 5.4 shows the disagreement cost of CMNs inferred with the A functions. Table 5.2 shows the disagreement count averaged over all of the experimental parameter values. This table shows that, on average, the disagreement count for the A functions is lower than for the P functions and that the fighter aircraft, on average, have a significantly higher disagreement count than the other classes. This provides more evidence for the same conclusions made from the entropy measurements.

Table 5.2: Average disagreement count

|         | P functions | A functions |
|---------|-------------|-------------|
| Single  | 14310       | 13209       |
| Twin    | 14729       | 13702       |
| Fighter | 26437       | 24251       |

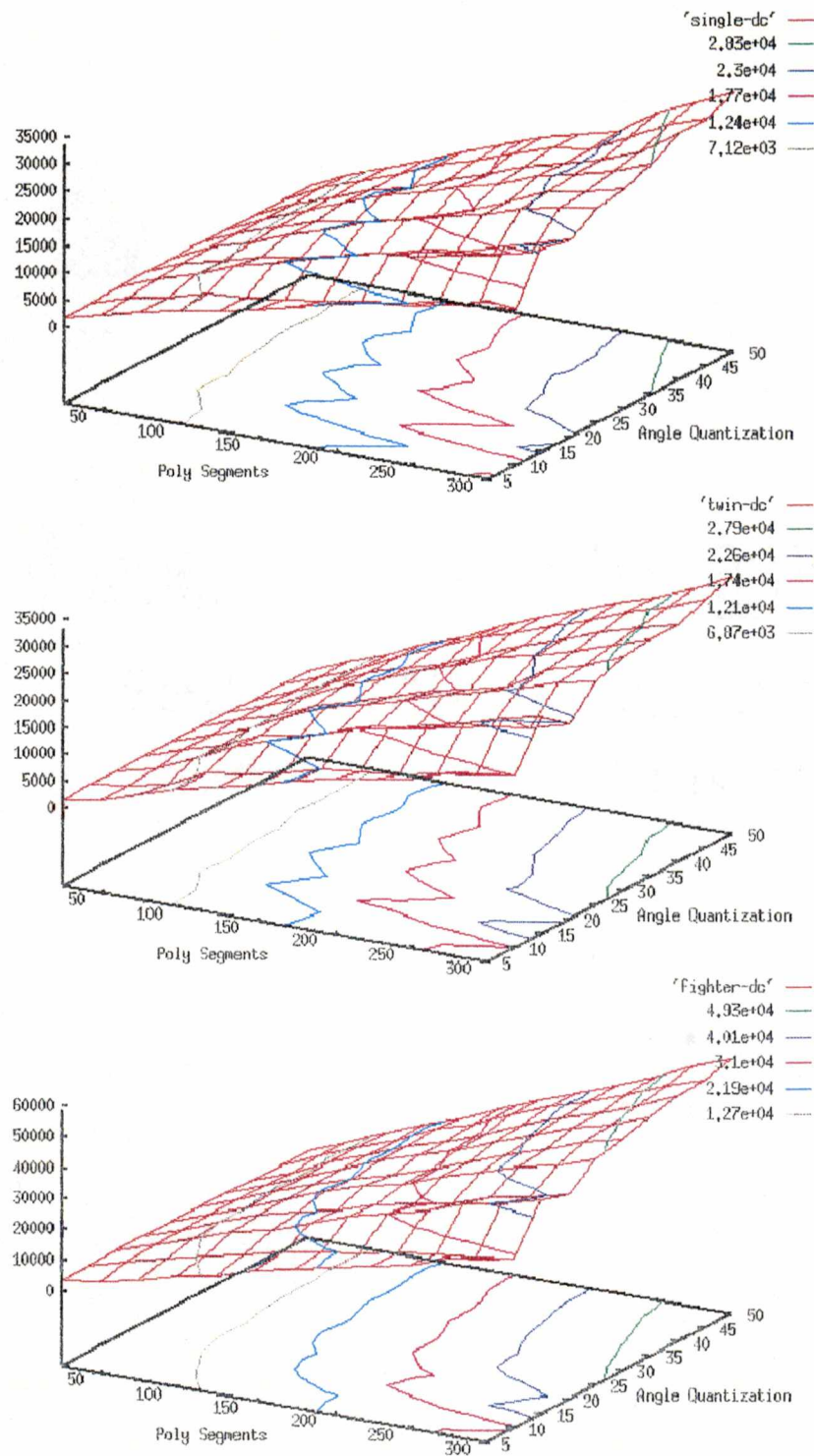


Figure 5.3: Disagreement count: Dynamic polygon segments, P functions

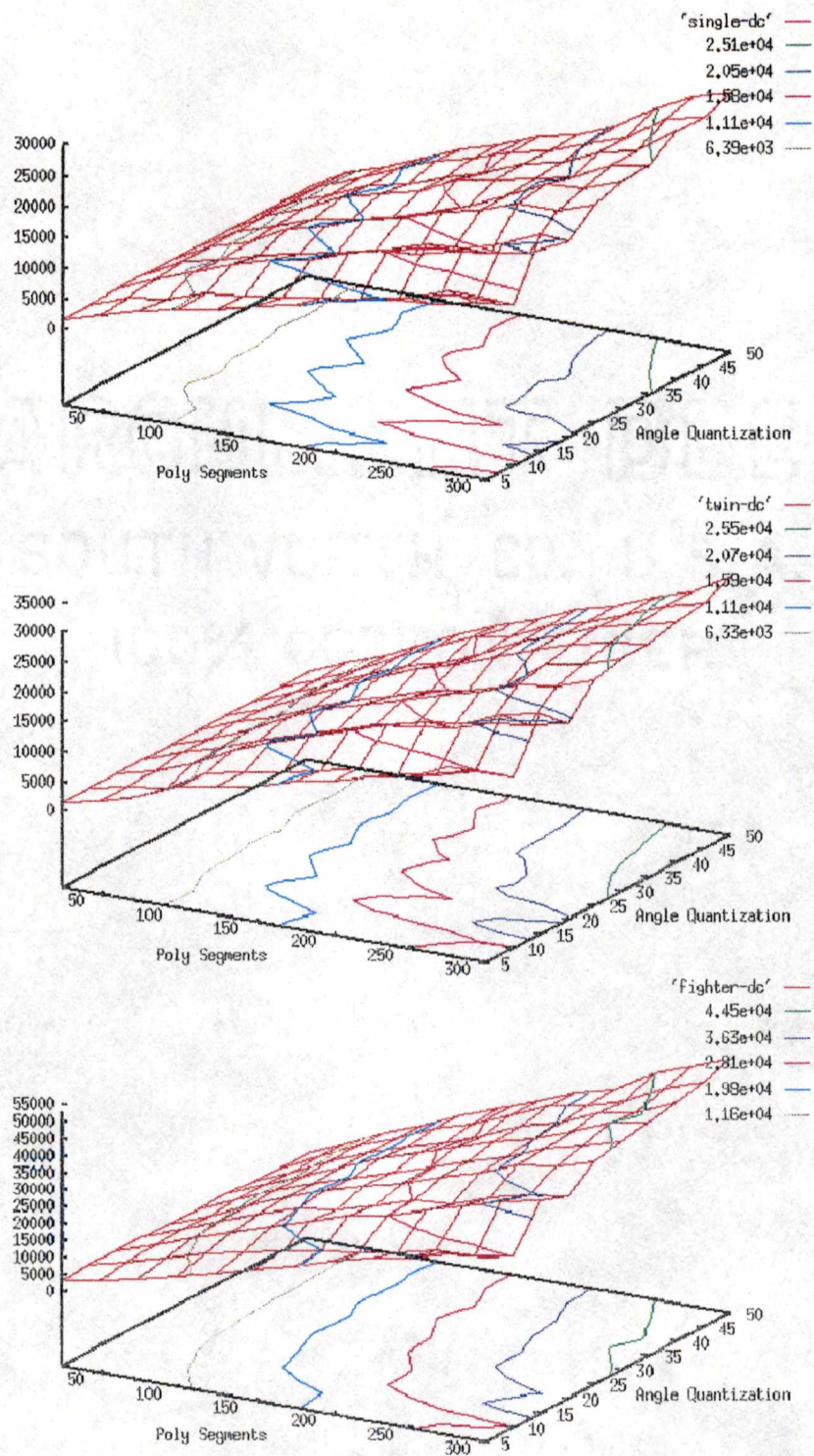


Figure 5.4: Disagreement count: Dynamic polygon segments, A functions

## Chapter 6

# Experimental Results

This chapter presents two sets of experimental results. The first set of results illustrates the classification performance of the method for a varied set of parameter values. The second set of results are rotational statistics gathered from the cyclical alignment algorithm for a subset of the parameters.

To reiterate about parameters, during experimentation the polygon segments were chosen in two different ways, either by a desired length (static segments) or a desired number of segments (dynamic segments) and the length or number of each of these were set to various values. Also set to various values were the number of angle quantization divisions. Experimentation was done with two different sets of probability functions: P functions and A functions.

## **6.1 Classification Performance**

Classification performance was studied using the aircraft data. Since the number of samples was small, 63, in each cycle of an experiment a single sample was used as the unknown sample and all of the other samples were grouped into their respective classes and used to train a CMN for each class. Next, a match was performed between the unknown sample and each of the CMNs so that the sample could be classified. These cycles were repeated until every sample was used once as the unknown sample. Tables 6.1, 6.2, 6.3, and 6.4 illustrate classification confusion for all combinations of probability functions and segment types.

### **6.1.1 Dynamic Segments vs. Static Segments**

Comparing tables 6.1 and 6.3 (static segments) with tables 6.2 and 6.4 (dynamic segments), show that although static segments occasionally result in better results, dynamic segments produce better results overall. Average percent classified correctly for all parameter values which include using static segments is 74.4 and for dynamic segments is 83.3. In particular, dynamic segments show much better results in recognition of the fighter aircraft, which have a much larger variance in number of contour points. It is theorized that this is true because the dynamic segments method results in a consistent quantity of values in the angle list for samples that have different quantities of contour points.

Table 6.1: Confusion matrix: Static polygon segments, P functions

|         | Min    |      |         | Avg    |      |         | Max    |       |         |
|---------|--------|------|---------|--------|------|---------|--------|-------|---------|
|         | Single | Twin | Fighter | Single | Twin | Fighter | Single | Twin  | Fighter |
| Single  | 53.7   | 0.0  | 0.0     | 97.4   | 2.6  | 0.0     | 100.0  | 46.2  | 0.0     |
| Twin    | 0.0    | 52.6 | 0.0     | 18.2   | 81.5 | 0.3     | 47.4   | 100.0 | 21.1    |
| Fighter | 8.0    | 0.0  | 0.0     | 55.0   | 11.1 | 33.9    | 88.0   | 36.0  | 92.0    |

Table 6.2: Confusion matrix: Dynamic polygon segments, P functions

|         | Min    |      |         | Avg    |      |         | Max    |       |         |
|---------|--------|------|---------|--------|------|---------|--------|-------|---------|
|         | Single | Twin | Fighter | Single | Twin | Fighter | Single | Twin  | Fighter |
| Single  | 52.6   | 0.0  | 0.0     | 92.8   | 6.4  | 0.8     | 100.0  | 47.4  | 15.8    |
| Twin    | 0.0    | 57.9 | 0.0     | 9.6    | 90.0 | 0.4     | 42.0   | 100.0 | 10.5    |
| Fighter | 0.0    | 0.06 | 16.0    | 20.6   | 15.0 | 64.4    | 60.0   | 64.0  | 96.0    |

Table 6.3: Confusion matrix: Static polygon segments, A functions

|         | Min    |      |         | Avg    |      |         | Max    |       |         |
|---------|--------|------|---------|--------|------|---------|--------|-------|---------|
|         | Single | Twin | Fighter | Single | Twin | Fighter | Single | Twin  | Fighter |
| Single  | 89.5   | 0.0  | 0.0     | 97.7   | 2.3  | 0.0     | 100.0  | 10.5  | 0.0     |
| Twin    | 0.0    | 68.4 | 0.0     | 11.6   | 88.4 | 0.0     | 31.6   | 100.0 | 0.0     |
| Fighter | 4.0    | 0.0  | 0.0     | 45.0   | 8.0  | 47.0    | 96.0   | 28.0  | 96.0    |

Table 6.4: Confusion matrix: Dynamic polygon segments, A functions

|         | Min    |      |         | Avg    |      |         | Max    |       |         |
|---------|--------|------|---------|--------|------|---------|--------|-------|---------|
|         | Single | Twin | Fighter | Single | Twin | Fighter | Single | Twin  | Fighter |
| Single  | 47.4   | 0.0  | 0.0     | 93.4   | 6.1  | 0.5     | 100.0  | 52.6  | 5.3     |
| Twin    | 0.0    | 52.6 | 0.0     | 9.8    | 89.5 | 0.7     | 42.1   | 100.0 | 21.1    |
| Fighter | 0.0    | 0.0  | 12.0    | 22.0   | 8.4  | 69.6    | 72.0   | 56.0  | 100.0   |

This will allow areas of the various aircraft that have similarities to show up in the same position in the training string, which will have the effect of compounding the likeliness of these “subsections” during training and matching.

### **6.1.2 Performance vs. Static Polygon Segment Length**

Graphs 6.1, 6.2, 6.3, and 6.4 show classification performance for various angle quantization divisions and polygon segment lengths (static segments) or segment quantities (dynamic segments). In all of these graphs, a pure white block is 100 percent and a pure black block is 80 percent or lower of the sample space that was classified correctly.

Studying graphs 6.1 and 6.3 reveals that performance vs. static polygon segment length is data dependent. For single engine aircraft samples, a segment length of 20 or more will give excellent results and a segment length as low as 5 will give good results. For fighter aircraft samples, a segment length of at least 30 is necessary to get passable results, and for twin engine aircraft samples, almost any segment length will give good results. It seems most likely that it is necessary to achieve some balance between short segment length, which preserves the most information and long segment length, which does not preserve as much information but helps to remove error obtained during the angle quantization process and some of the diversity of the samples.

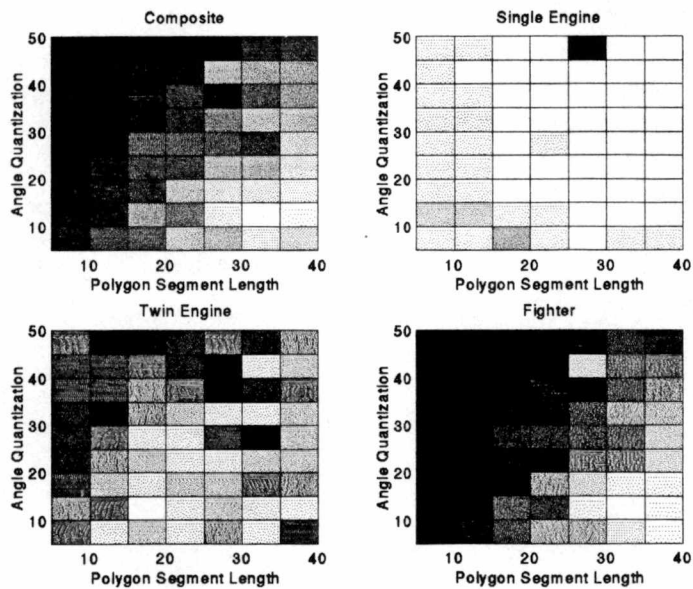


Figure 6.1: Correctly classified: Static polygon segments, P functions

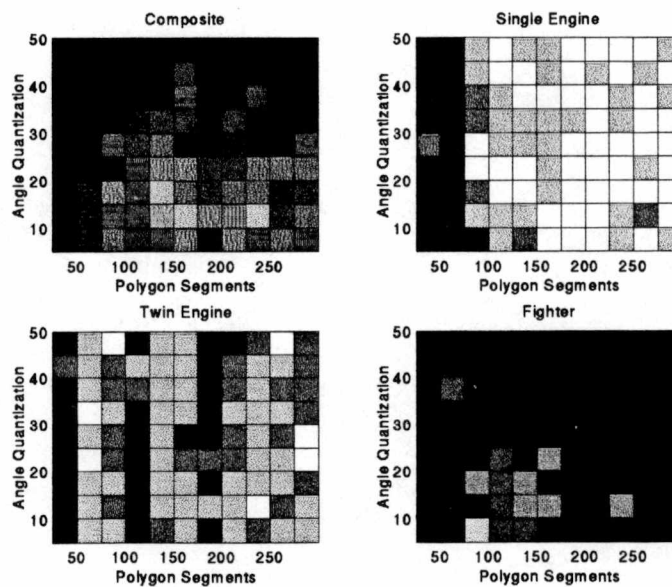


Figure 6.2: Correctly classified: Dynamic polygon segments, P functions

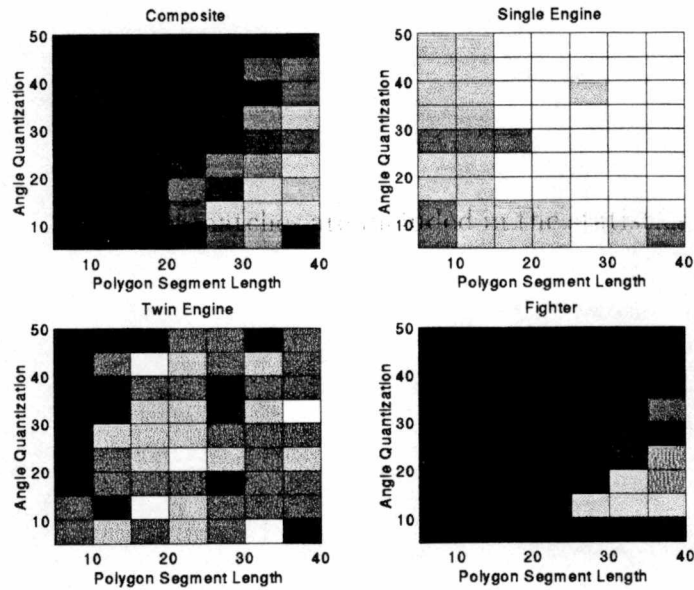


Figure 6.3: Correctly classified: Static polygon segments, A functions

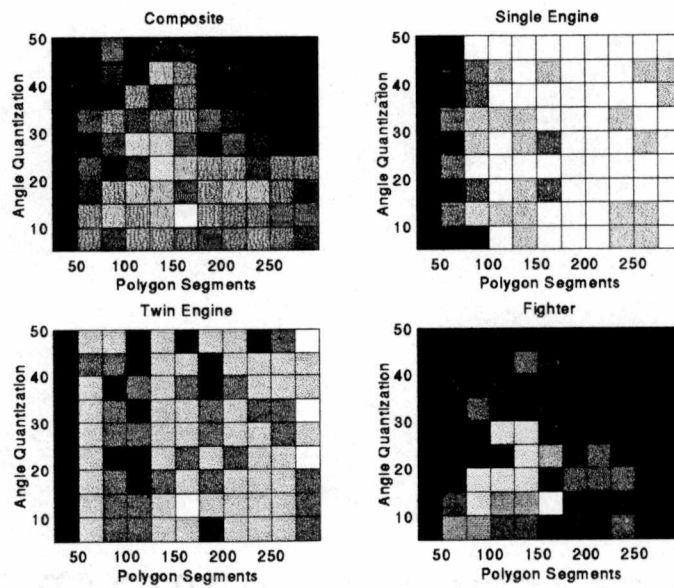


Figure 6.4: Correctly classified: Dynamic polygon segments, A functions

As to why the fighter aircraft recognition performance is improved with greater segment length: long segment length is able to overcome the diversity of shape seen in the fighter aircraft, while retaining enough of their attributes to make them recognizable. Both single and twin engine aircraft have a consistency of shape within their own class, therefore variations of segment length do not have as much effect of their recognizability.

### **6.1.3 Performance vs. Dynamic Polygon Segment Quantity**

Studying graphs 6.2 and 6.4 reveals that performance vs. dynamic polygon segment quantity is also data dependent. The quantity of segments when using dynamic segments does not have as pronounced an effect on performance as the segment length when using static segments. The fighter aircraft samples have the narrowest range of segment quantity that result in good recognition performance. Anything between 50 and 175 segments per polygon is preferred. The reason for this parallels what was discussed in the previous section. It is necessary to have a quantity of segments large enough to overcome the diversity of a single class of samples while having a small enough number to still retain some recognizable information.

#### **6.1.4 Performance vs. Angle Quantization Divisions**

All of the graphs in this chapter point out that the number of divisions of the angle quantization have only a small effect on recognition performance as long as between 5 and 80 divisions are used. Quantities of divisions in this range suppress diversity and noise in the samples to allow for better recognizability. Also, it is notable that a smaller number of divisions allows for faster computation during inference and matching.

#### **6.1.5 P Probability Functions vs. A Probability functions**

Consulting tables 6.1, 6.2, 6.3, and 6.4 shows that even though the P functions occasionally perform better for one class or another, the A functions perform more consistently, particularly for the hard to recognize fighter aircraft. Average percent of samples classified correctly for all parameter values including use of P functions is 77 and for A functions is 81.

As briefly discussed earlier, the A functions do not take into account that the training strings are lists of angles. The P functions take this into account by making two angles that are farther apart a less likely match than two angles that are close together. Hence, this is the reason for the overall improved classification performance when using the A functions.

## 6.2 Rotational Statistics

Tables 6.5, 6.6, and 6.7 show mean rotational error statistics obtained during the cyclical match for single engine aircraft, twin engine aircraft, and fighter aircraft, respectively. Only successful matches are included in the statistics and an asterisk (\*) denotes a set of parameters that did not result in any successful matches. These statistics were obtained using dynamic segments and the A probability functions. In these tables the rows, labeled with D, represent the divisions of angle quantization and the columns, labeled with Q, represent the number of polygon segments.

Surprisingly the fighter aircraft samples are less susceptible to rotational error. Studying these statistics, along with the classification performance shows that when a fighter aircraft is classified correctly the rotational error is very close to 0 and when they are misclassified the rotational error is large. It is theorized that because most of the fighter aircraft have sharp corners it is easy to find the correct starting point. The single engine and twin engine rotational error statistics show that the amount of rotational error increases as the number of segments of the polygon increases. Single and twin engine aircraft silhouettes are characterized by smooth, rounded corners, so as the number of polygon segments is increased in their representation, it is much harder to find the actual starting point.

Table 6.5: Single engine aircraft: Mean rotational error (in segments)

|    | Q | 25  | 50  | 75  | 100 | 125  | 150  | 175  | 200  | 225  | 250  | 275  | 300  |
|----|---|-----|-----|-----|-----|------|------|------|------|------|------|------|------|
| D  |   |     |     |     |     |      |      |      |      |      |      |      |      |
| 5  |   | 1.8 | 1.4 | 8.6 | 5.7 | 8.0  | 25.3 | 22.0 | 12.6 | 42.1 | 34.7 | 43.6 | 61.2 |
| 10 |   | 1.5 | 1.1 | 8.8 | 3.7 | 13.0 | 16.1 | 14.3 | 17.0 | 52.2 | 28.6 | 49.5 | 35.5 |
| 15 |   | 1.6 | 3.5 | 4.5 | 3.6 | 10.0 | 20.8 | 10.7 | 21.4 | 23.9 | 33.4 | 43.0 | 42.7 |
| 20 |   | 2.0 | 4.0 | 8.2 | 3.5 | 7.2  | 8.7  | 22.6 | 12.3 | 44.7 | 44.0 | 74.1 | 50.5 |
| 25 |   | 1.4 | 1.2 | 8.9 | 3.5 | 7.6  | 13.4 | 19.0 | 21.3 | 34.3 | 47.2 | 43.5 | 43.3 |
| 30 |   | 2.3 | 1.7 | 7.3 | 5.8 | 10.2 | 8.7  | 26.1 | 26.0 | 46.8 | 50.0 | 55.5 | 28.8 |
| 35 |   | 2.7 | 4.3 | 8.0 | 3.5 | 7.2  | 11.9 | 18.9 | 26.0 | 44.5 | 56.2 | 52.6 | 42.4 |
| 40 |   | 2.3 | 1.3 | 8.6 | 5.8 | 10.0 | 16.0 | 20.7 | 21.4 | 39.7 | 42.4 | 45.4 | 50.6 |
| 45 |   | 2.1 | 1.2 | 6.6 | 5.5 | 7.2  | 8.8  | 10.7 | 21.4 | 44.3 | 50.4 | 59.1 | 35.6 |
| 50 |   | 2.3 | 3.0 | 8.2 | 6.0 | 15.0 | 17.7 | 14.5 | 45.4 | 19.3 | 46.8 | 67.7 | 35.7 |

Table 6.6: Twin engine aircraft: Mean rotational error (in segments)

|    | Q | 25  | 50  | 75  | 100  | 125  | 150  | 175  | 200  | 225  | 250  | 275  | 300  |
|----|---|-----|-----|-----|------|------|------|------|------|------|------|------|------|
| D  |   |     |     |     |      |      |      |      |      |      |      |      |      |
| 5  |   | 0.8 | 1.4 | 9.6 | 20.2 | 13.3 | 16.8 | 22.0 | 26.9 | 30.2 | 22.7 | 48.1 | 65.1 |
| 10 |   | 0.9 | 1.9 | 9.7 | 16.4 | 12.6 | 15.8 | 16.1 | 32.1 | 31.1 | 47.3 | 42.4 | 53.0 |
| 15 |   | 1.6 | 1.2 | 7.2 | 12.6 | 14.4 | 12.6 | 20.0 | 31.7 | 35.8 | 30.5 | 45.2 | 41.6 |
| 20 |   | 1.0 | 1.2 | 4.3 | 7.0  | 13.2 | 17.1 | 23.1 | 23.7 | 32.3 | 51.7 | 38.2 | 47.9 |
| 25 |   | 0.6 | 1.2 | 5.4 | 9.0  | 9.8  | 16.8 | 32.2 | 27.7 | 36.1 | 54.3 | 67.6 | 46.8 |
| 30 |   | 1.2 | 1.5 | 5.5 | 10.8 | 8.2  | 19.4 | 20.1 | 27.8 | 38.7 | 53.9 | 39.3 | 61.1 |
| 35 |   | 1.5 | 2.0 | 4.0 | 5.4  | 5.0  | 13.5 | 18.5 | 24.2 | 31.5 | 29.5 | 31.4 | 34.2 |
| 40 |   | 1.5 | 1.7 | 5.7 | 9.7  | 5.6  | 19.5 | 14.2 | 28.2 | 31.2 | 45.5 | 47.0 | 49.2 |
| 45 |   | 1.7 | 1.7 | 6.0 | 13.0 | 7.5  | 14.7 | 19.6 | 27.2 | 46.5 | 24.6 | 37.2 | 46.5 |
| 50 |   | 1.2 | 2.1 | 4.2 | 7.1  | 8.3  | 22.8 | 13.4 | 27.3 | 31.9 | 47.3 | 65.7 | *    |

Table 6.7: Fighter engine aircraft: Mean rotational error (in segments)

|    | Q | 25  | 50  | 75  | 100 | 125 | 150 | 175 | 200 | 225  | 250  | 275  | 300  |
|----|---|-----|-----|-----|-----|-----|-----|-----|-----|------|------|------|------|
| D  |   |     |     |     |     |     |     |     |     |      |      |      |      |
| 5  |   | 1.0 | 2.1 | 1.0 | 5.5 | 5.8 | 2.5 | 3.2 | 9.3 | 13.3 | 12.1 | 20.5 | 23.2 |
| 10 |   | 1.3 | 2.0 | 1.9 | 2.7 | 1.9 | 2.1 | 0.5 | 7.2 | 7.0  | 4.9  | 10.4 | 13.2 |
| 15 |   | 2.5 | 1.3 | 0.8 | 2.3 | 1.8 | 0.0 | 2.5 | 5.1 | 6.6  | 11.9 | 11.3 | 11.9 |
| 20 |   | 2.0 | 0.7 | 0.0 | 2.0 | 0.0 | 0.0 | 2.8 | 6.3 | 5.1  | 4.9  | 6.1  | 8.6  |
| 25 |   | 1.2 | 1.3 | 0.0 | 1.6 | 0.0 | 0.0 | 3.4 | 3.1 | 7.3  | 0.0  | 0.0  | 8.6  |
| 30 |   | 1.7 | 0.7 | 1.0 | 3.6 | 0.0 | 2.5 | 3.6 | 4.4 | 9.5  | 0.0  | 7.7  | 7.0  |
| 35 |   | 1.3 | 0.7 | 0.1 | 1.5 | 0.0 | 0.0 | 7.6 | 5.0 | 0.0  | 0.0  | 11.1 | 21.3 |
| 40 |   | 0.8 | 0.7 | 0.0 | 1.7 | 0.0 | 0.0 | 7.1 | 0.2 | 0.0  | 5.3  | 0.0  | 27.6 |
| 45 |   | 2.8 | 1.4 | 1.1 | 0.0 | 0.0 | 0.0 | 4.5 | 6.4 | 0.0  | 9.5  | 0.0  | *    |
| 50 |   | 2.8 | 0.7 | 0.1 | 2.9 | 0.0 | 2.8 | 3.6 | 5.0 | 10.6 | 0.0  | 0.0  | *    |

As long as the rotational error is less than 10 percent of the quantity of polygon segments, it does not have a large effect on recognizability. This recognition technique applied to any samples with smooth corners would result in higher rotational error than if angular samples were used.

## Chapter 7

# Conclusion

The objective of this work was to apply well known techniques and algorithms to find a solution to various pattern classification problems. All of the algorithms comprising this method were presented along with reference to further information for most. The method was tested with sample data consisting of three classes of aircraft: single engine, twin engine, and fighter aircraft. Even though the sample space was small, the results obtained during trial use of this pattern classification method illustrate that the objective was achieved.

The technique presented herein is able to classify the aircraft samples correctly at least 90 percent of the time, given that a close to optimal set of parameters was chosen for the preprocessing. Even though a large span of parameter values results in good recognition performance, the optimal parameter choices proved to be data dependent. Future work should include an addition to this method

that allows the preprocessing parameters to be chosen automatically by looking at some feature, or features, in the data.

## Bibliography

# Bibliography

- [1] J. Gregor and M. G. Thomason. Efficient dynamic programming alignment of cyclic strings by shift elimination. *accepted for publication in Pattern Recognition*, 1995.
- [2] J. Gregor, M. G. Thomason, and C. Guthrie. Dynamic programming inference of constrained markov networks. Technical report, University of Tennessee, Knoxville, TN, August 1993.
- [3] C. Guthrie, J. Gregor, and M. G. Thomason. Constrained markov networks for automated analysis of g-banded chromosomes. *Comput. Biol. Med.*, 23:105–114, 1993.
- [4] M. Lambert, editor. *Jane's All The World's Aircraft*. Jane's Information Group, Surrey, UK, 1991.
- [5] M. Maes. On a cyclic string-to-string correction problem. *Information Processing Letters*, 35:73–78, 1990.

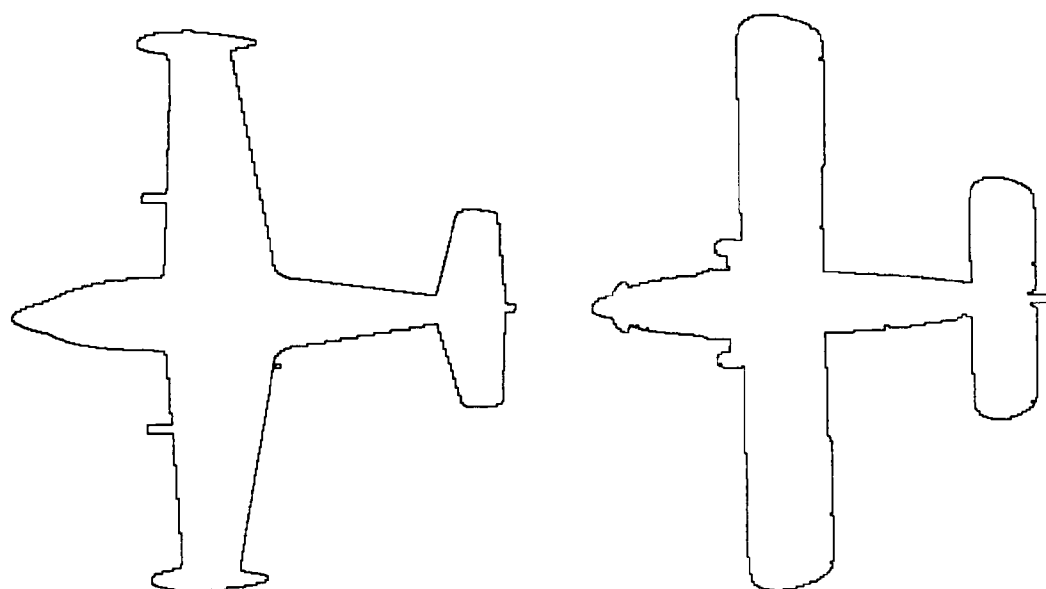
- [6] M. Maes. Polygonal shape recognition using string matching techniques. *Pattern Recognition Soc.*, 24:433–440, 1991.
- [7] F. Rannou. Equilateral polygon approximation of closed contours. Master's thesis, The University of Knoxville, Tennessee, Knoxville, TN, 1993.
- [8] F. Rannou and J. Gregor. Equilateral polygon approximation of closed contours. *accepted for publication in Pattern Recognition*, 1995.
- [9] R. Schalkoff. *Pattern Recognition*. John Wiley and Sons, Inc., 1992.
- [10] R. Sedgewick. *Algorithms in C++*. Addison-Wesley Publishing Company, 1992.

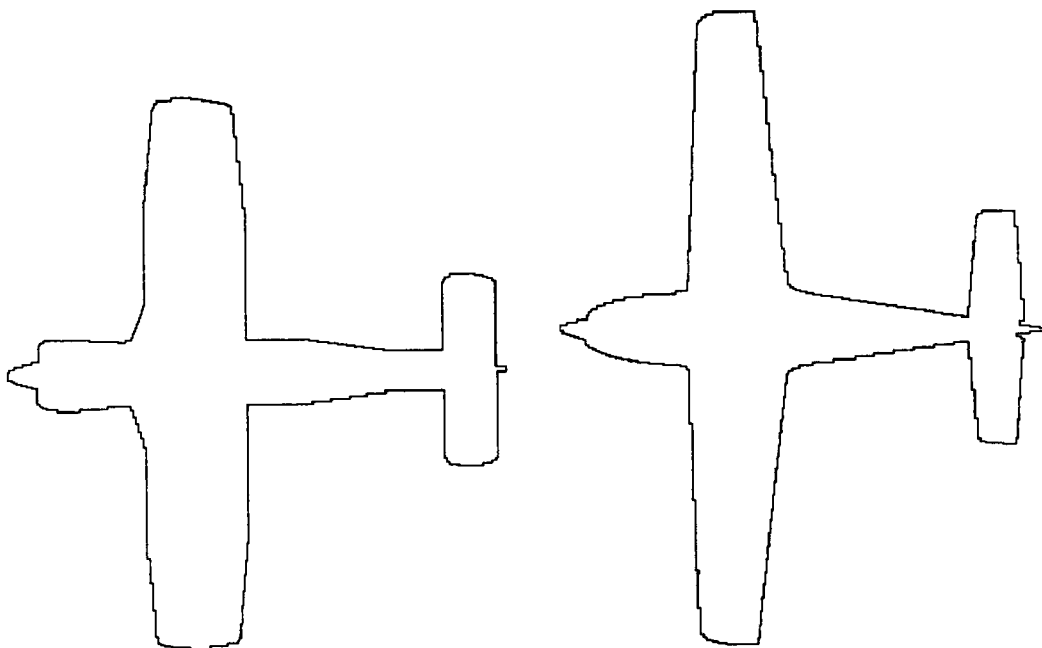
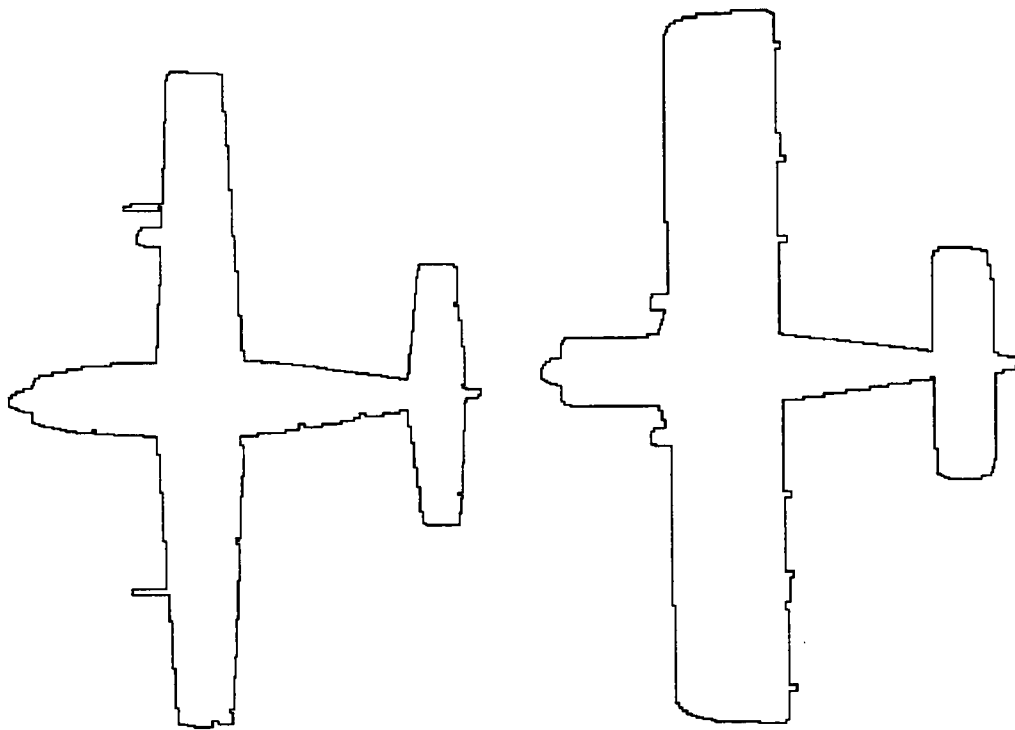
# Appendices

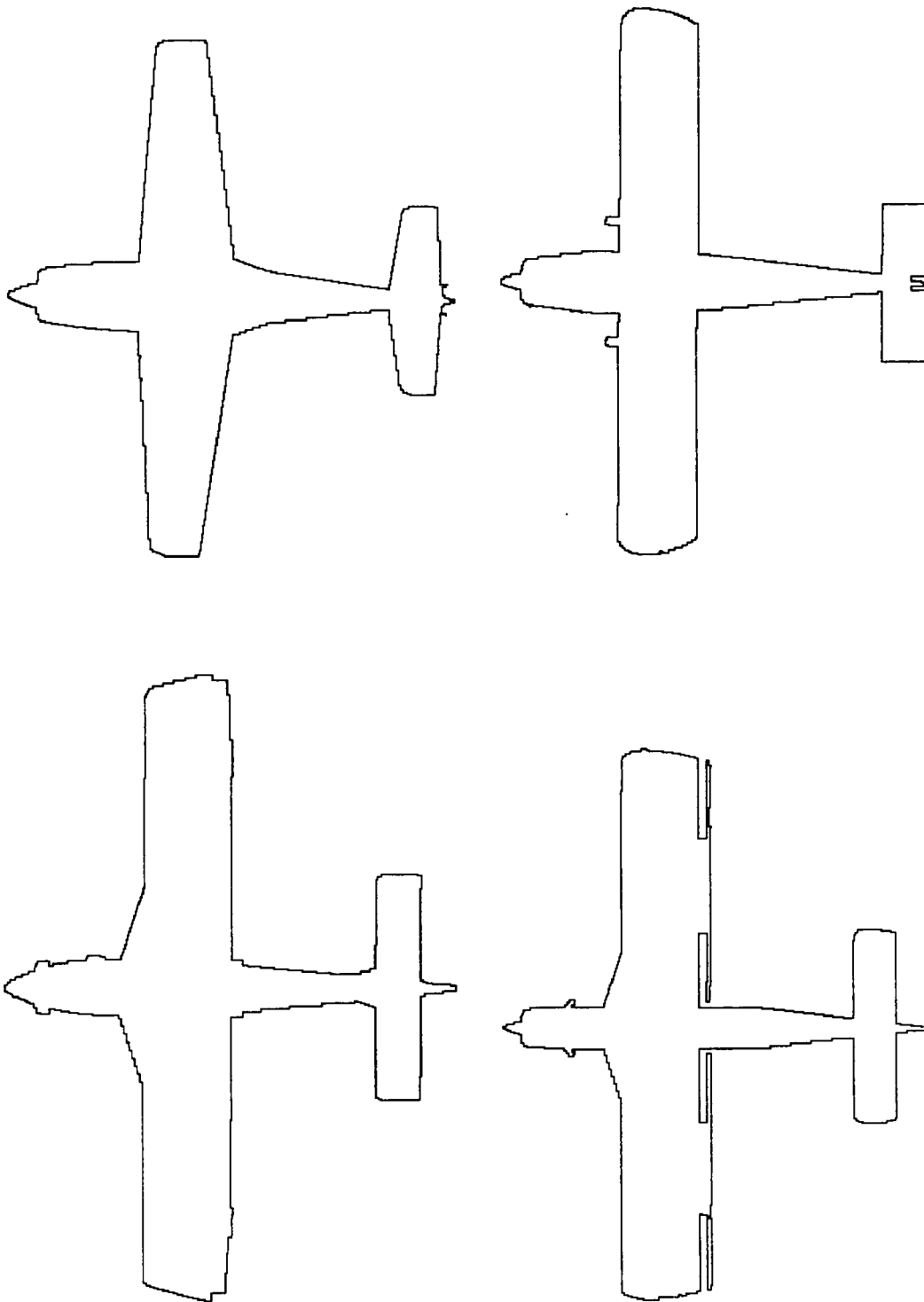
## Appendix A

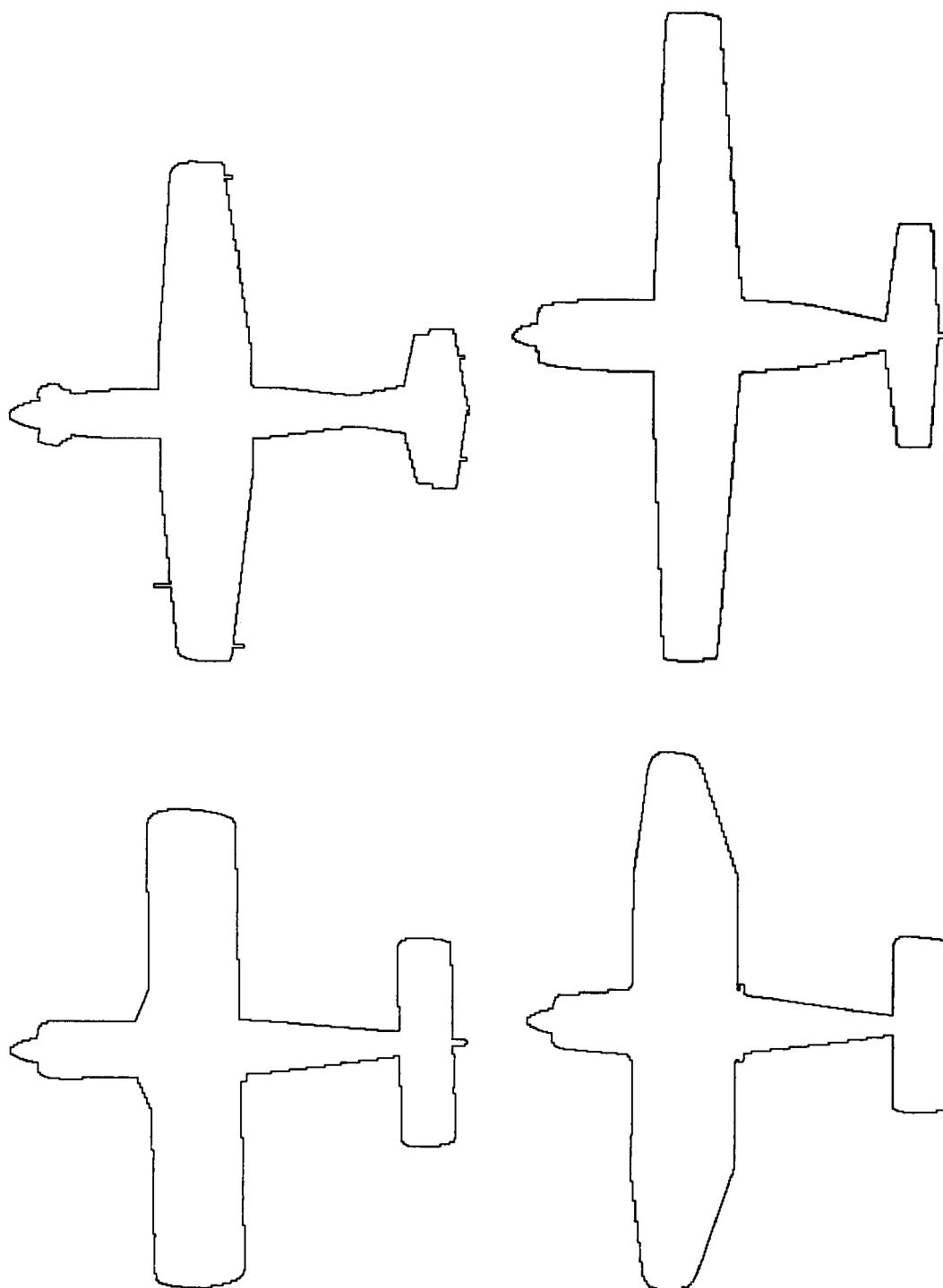
### Single Engine Aircraft Sample

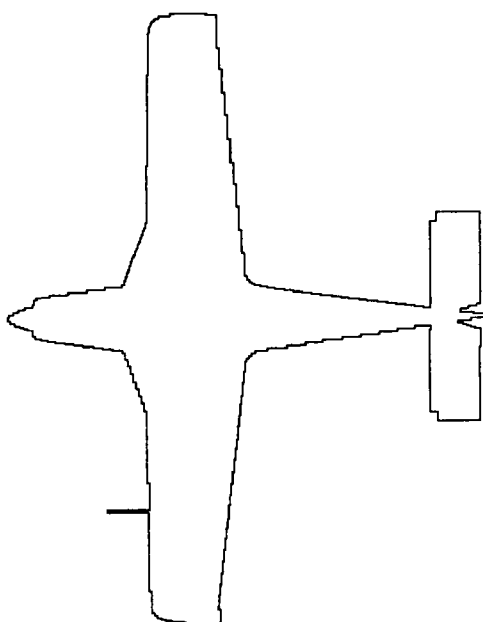
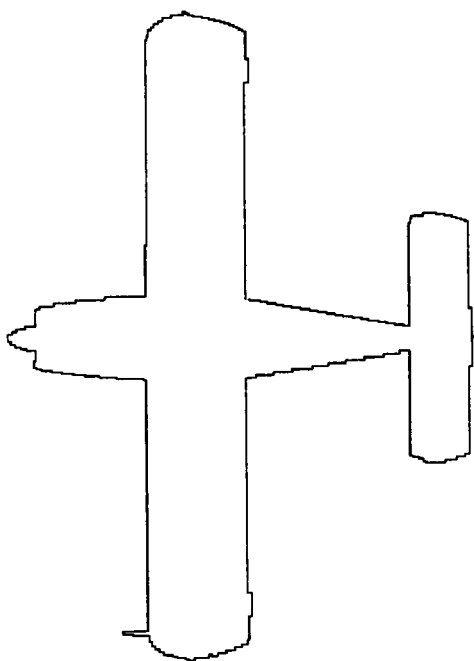
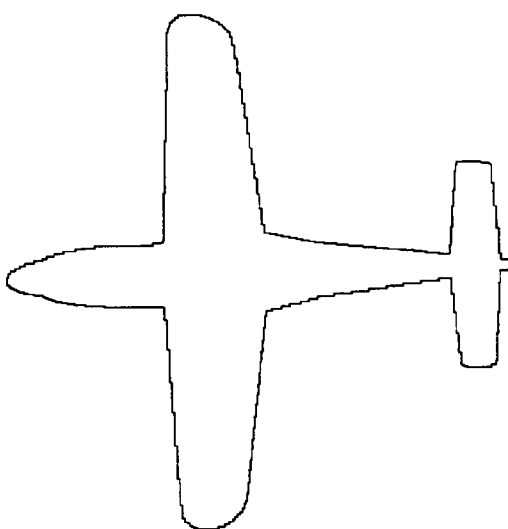
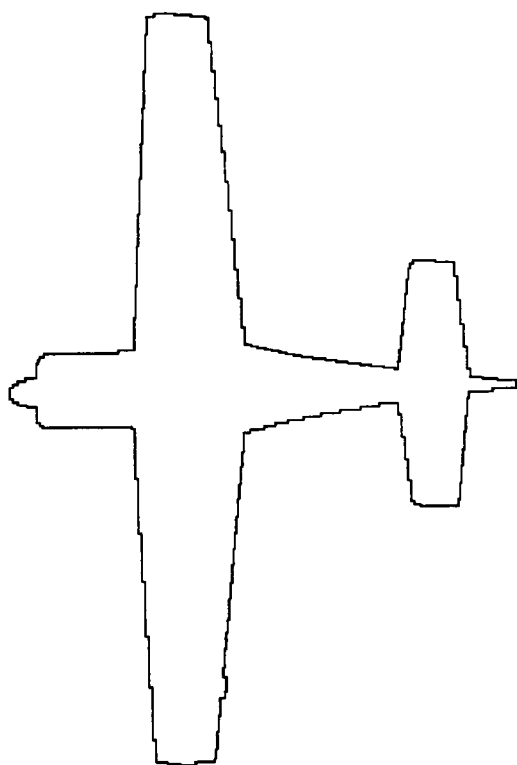
#### Contours

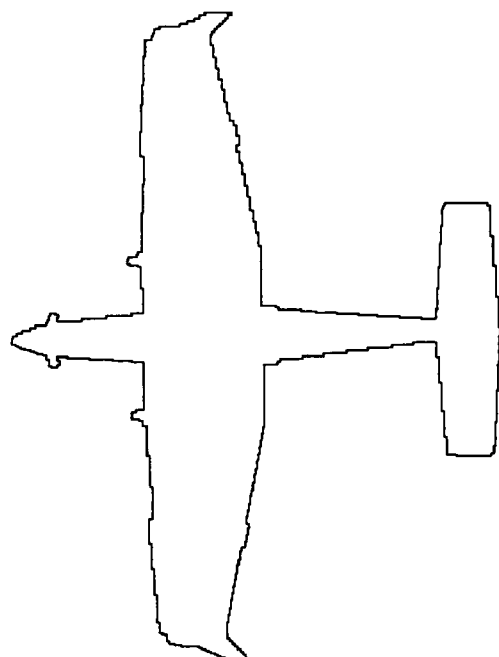








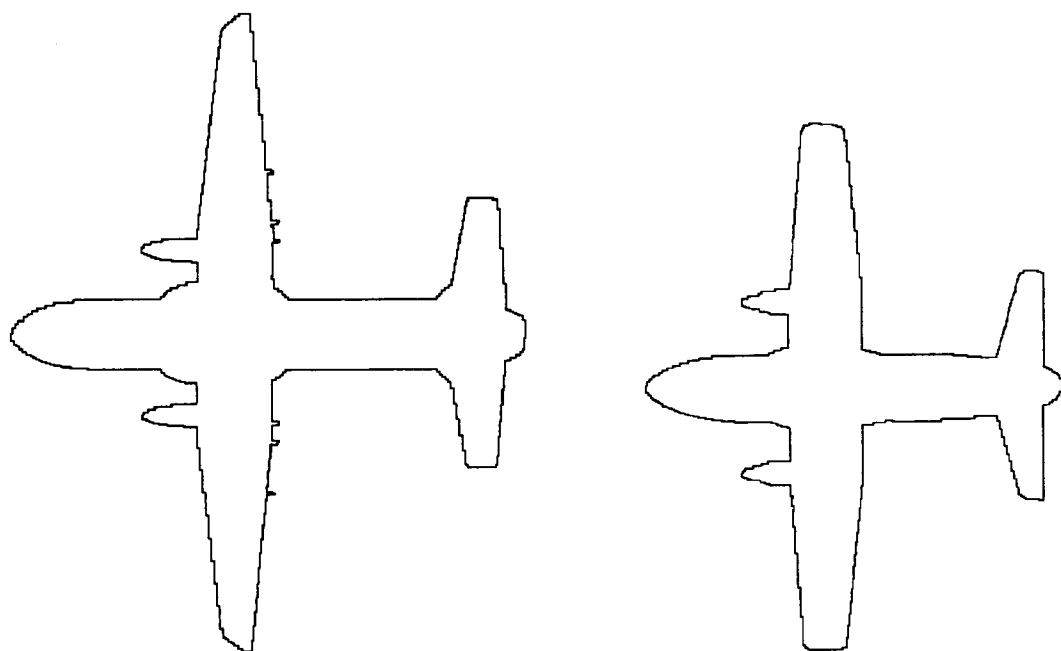


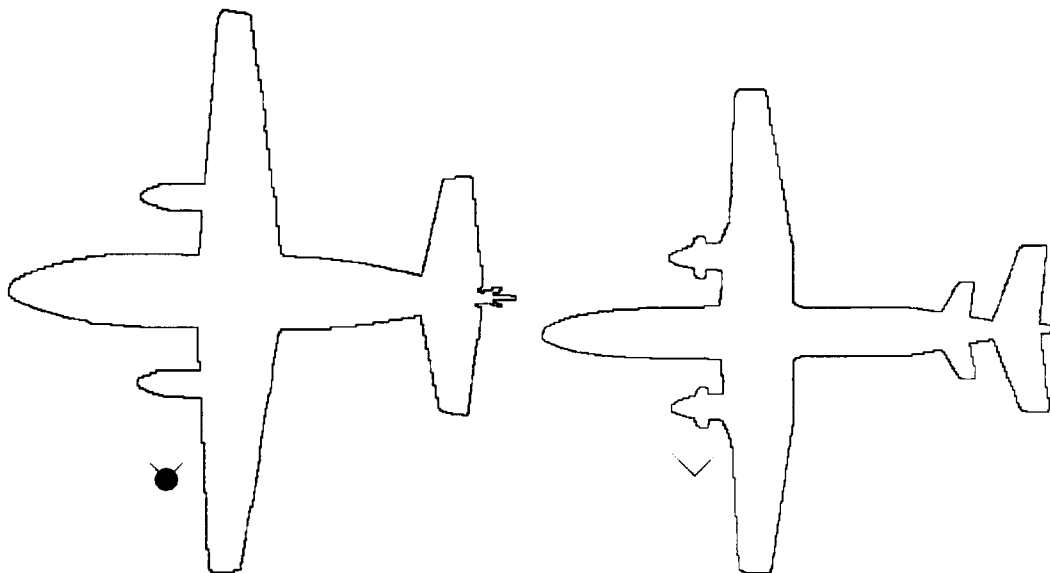
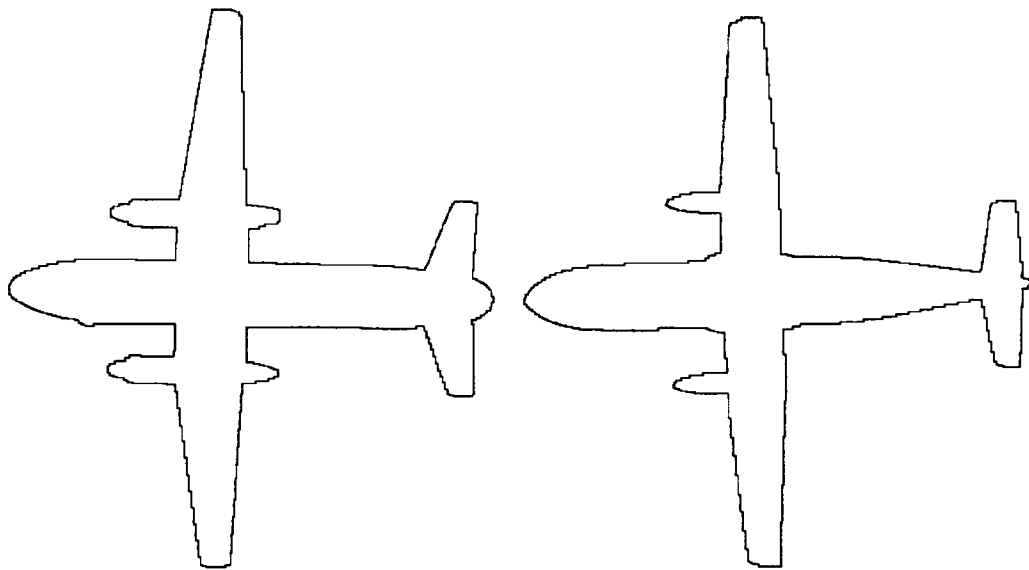


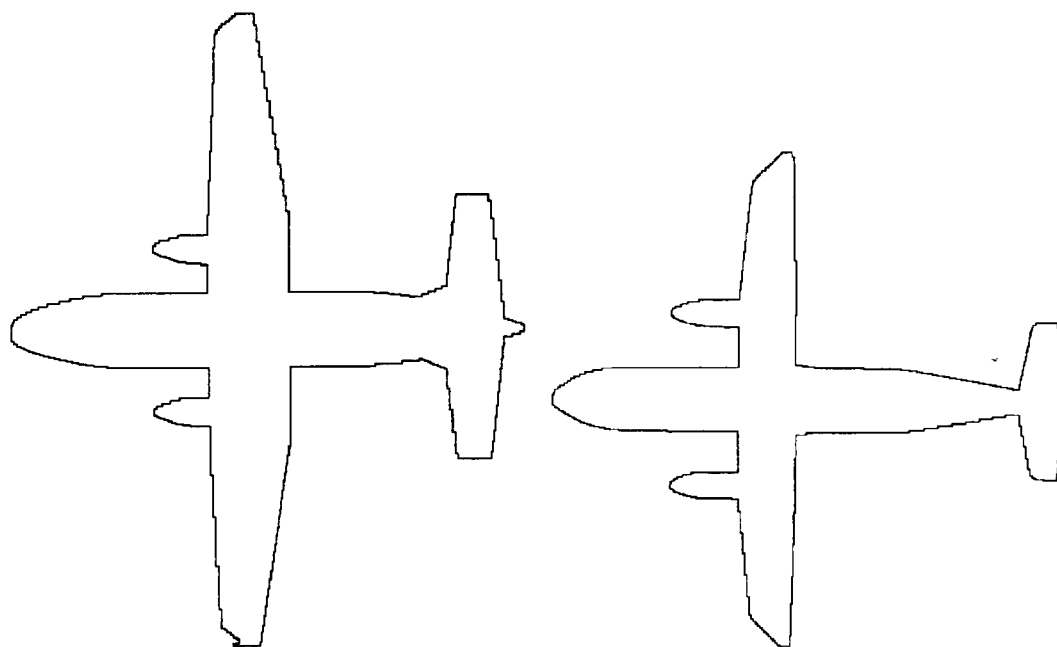
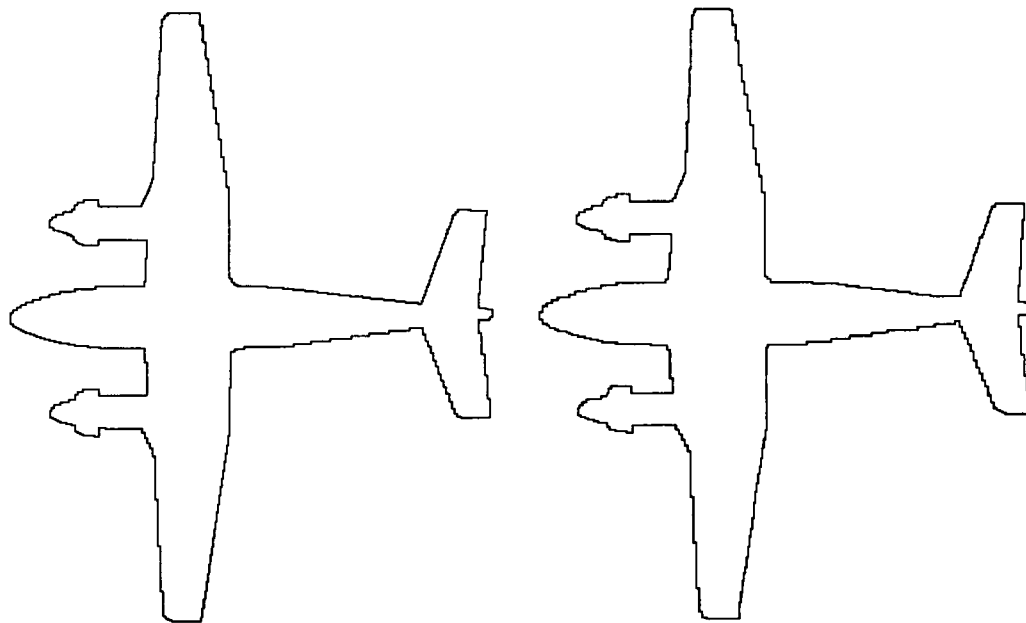
## Appendix B

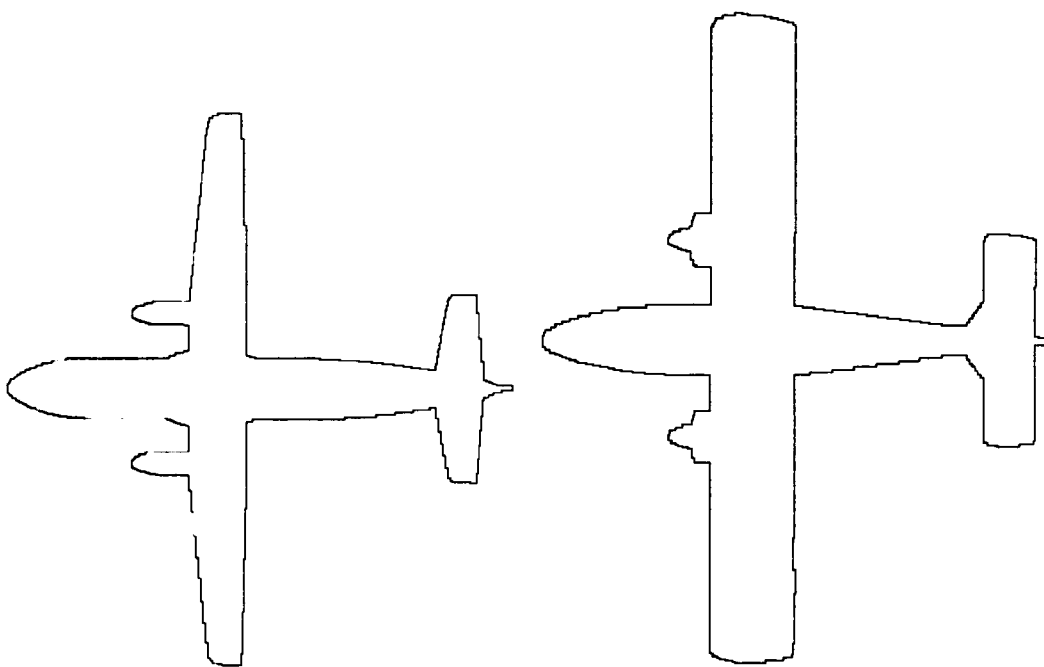
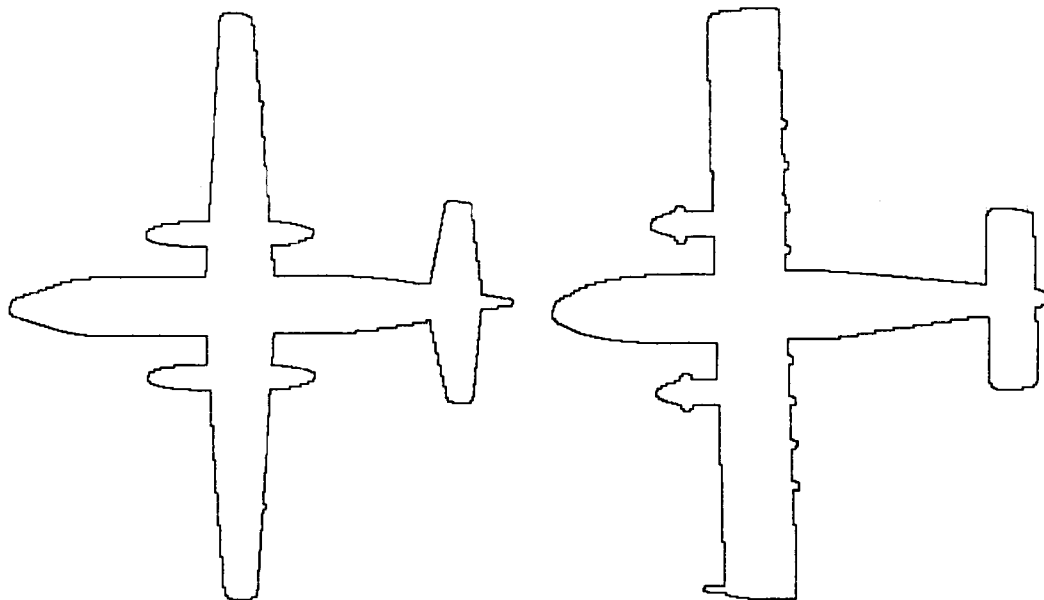
### Twin Engine Aircraft Sample

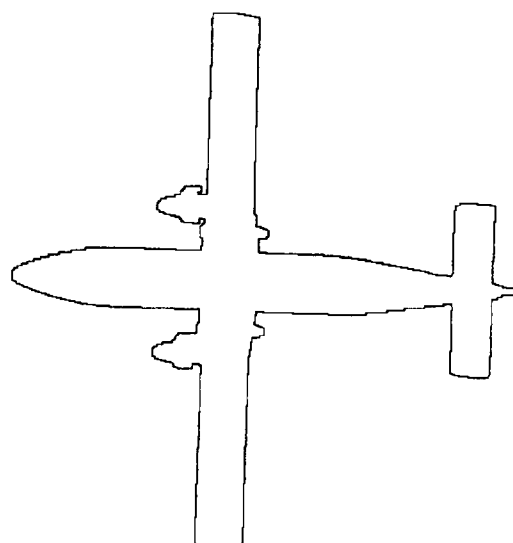
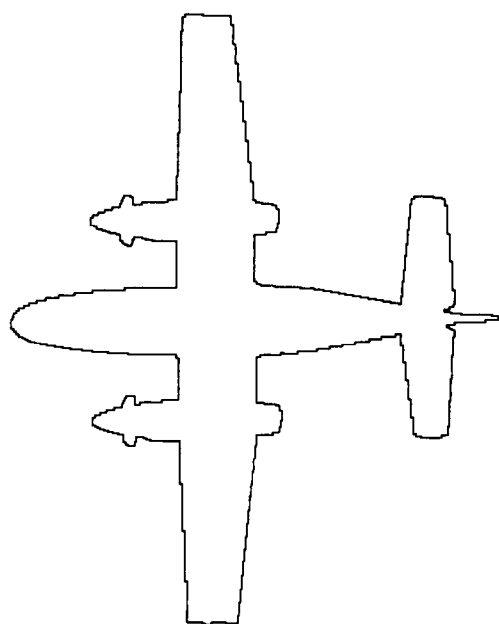
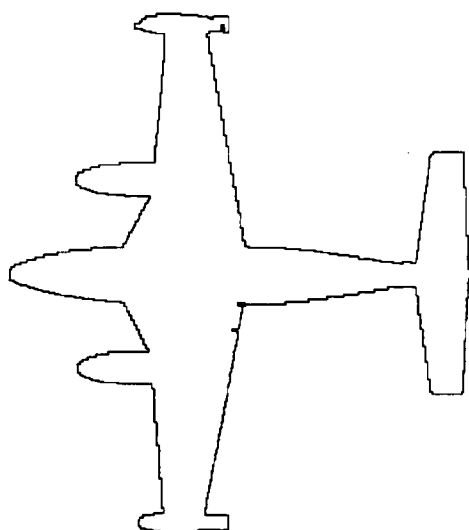
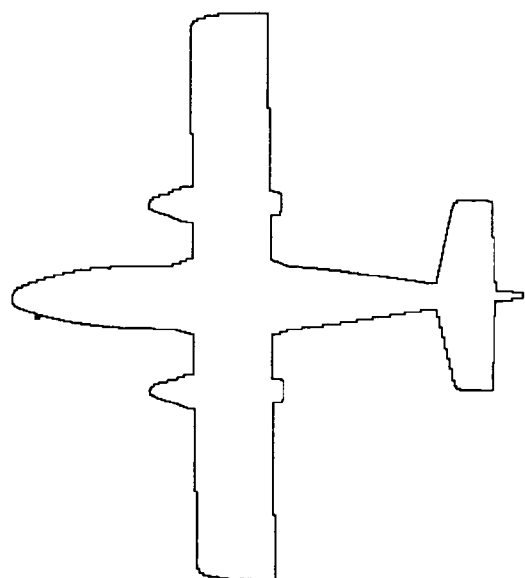
#### Contours

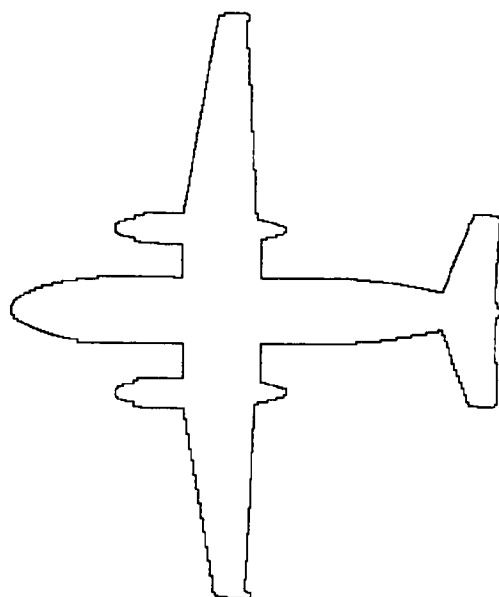






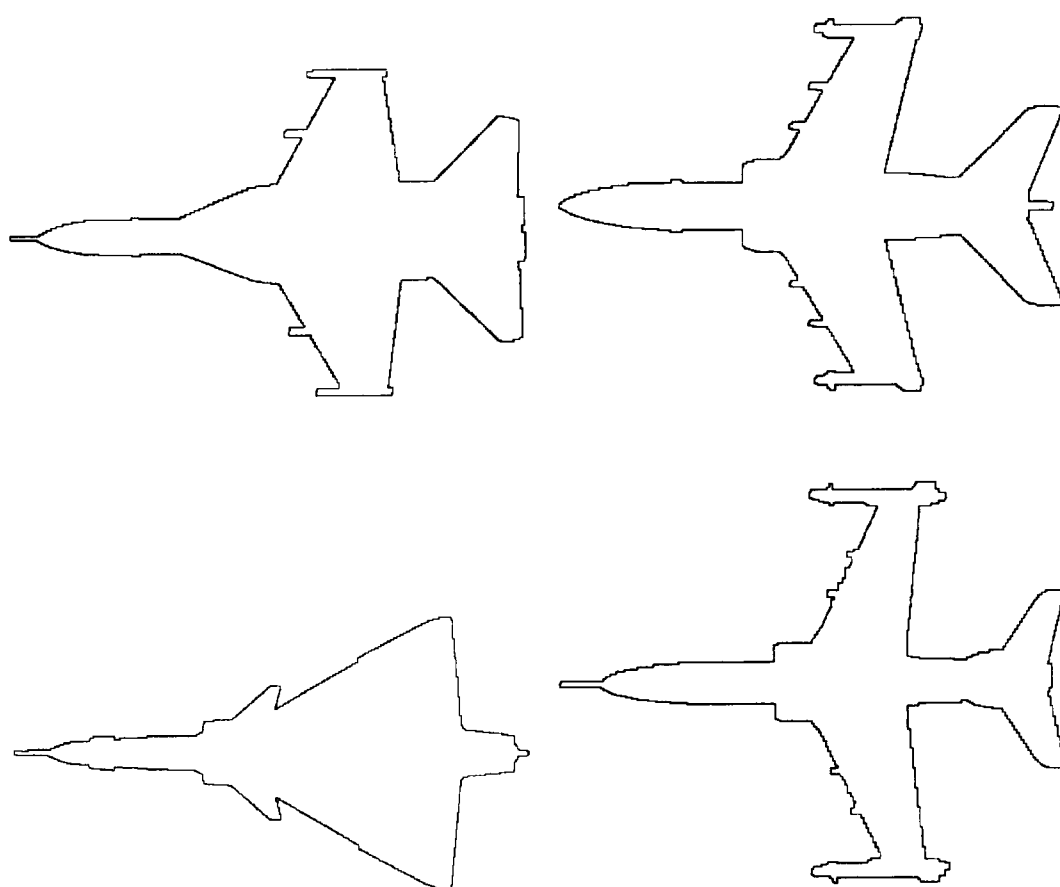


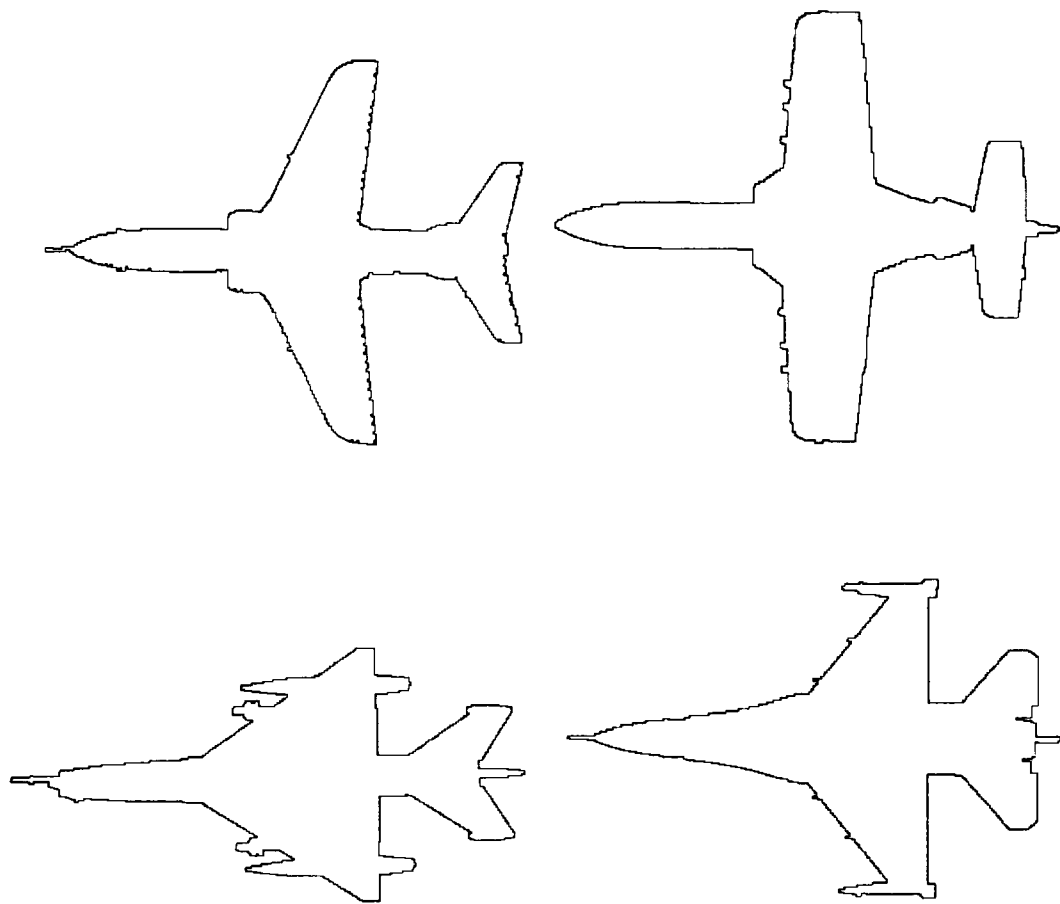


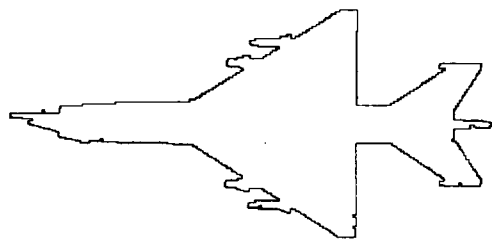
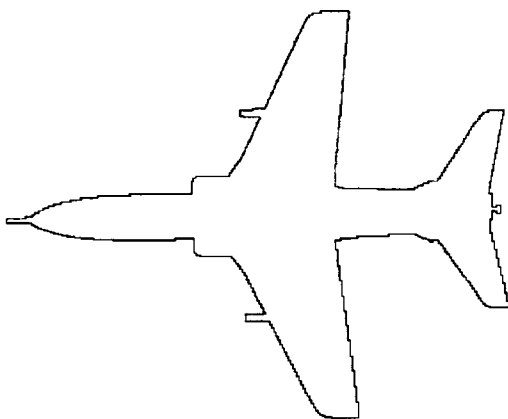
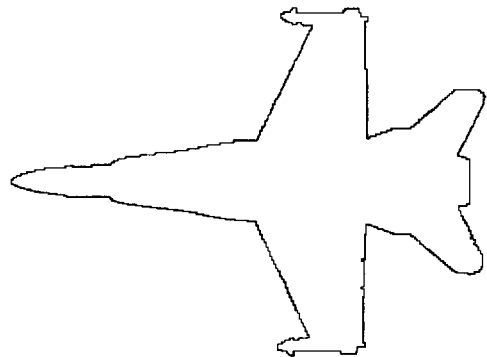
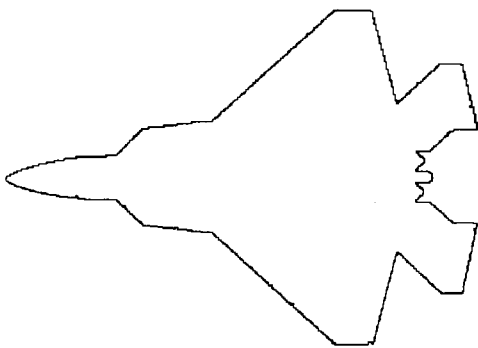
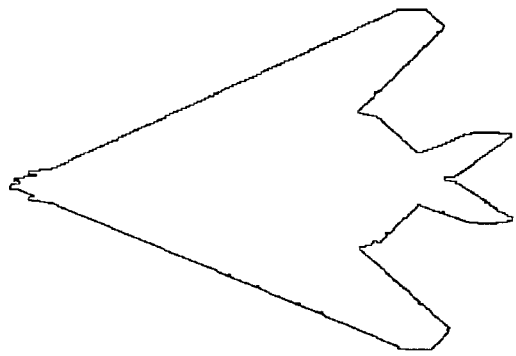
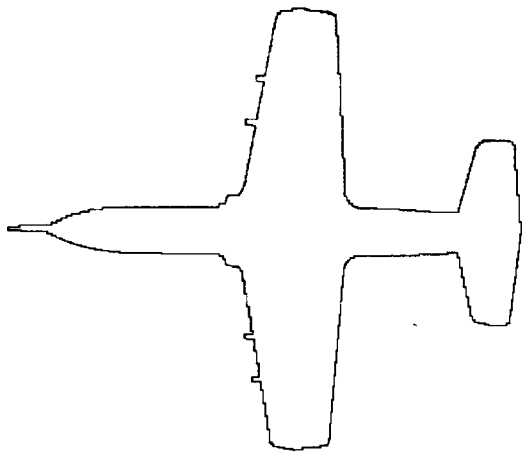


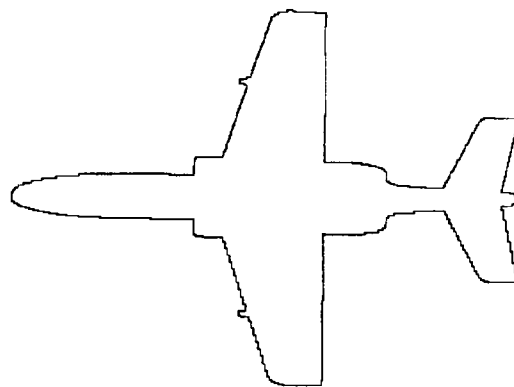
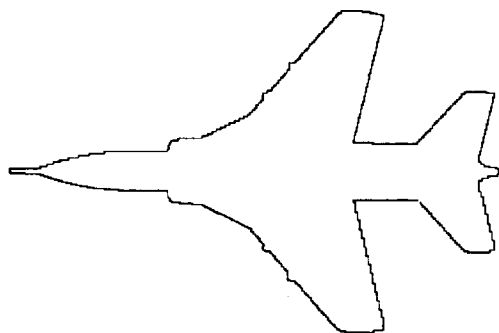
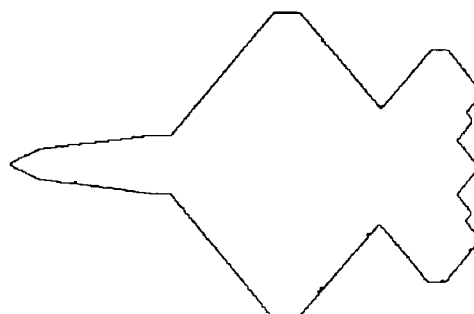
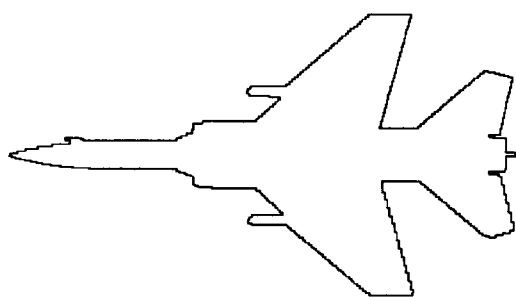
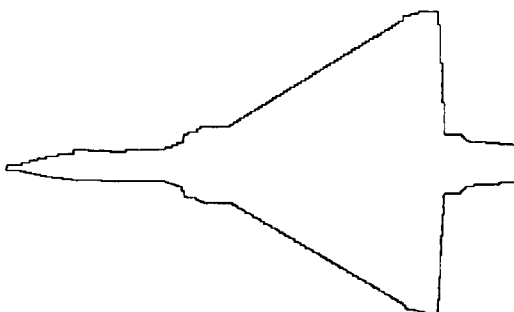
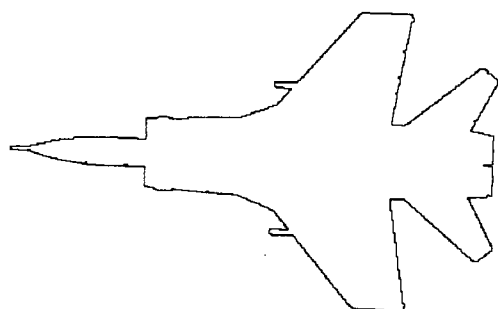
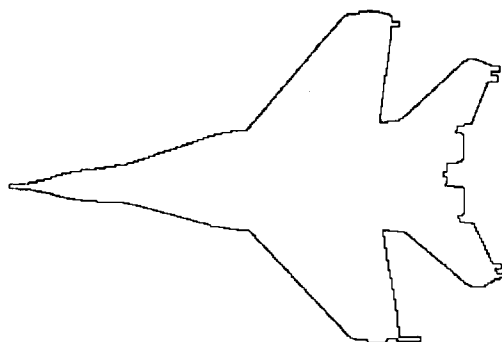
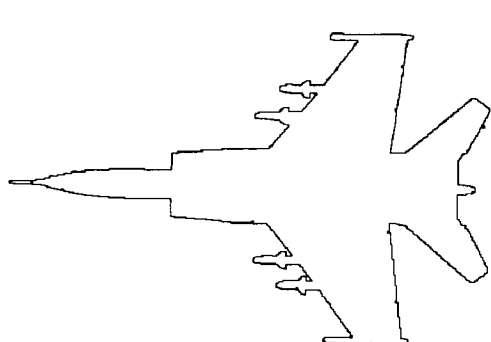
## Appendix C

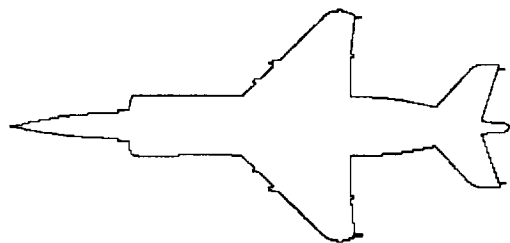
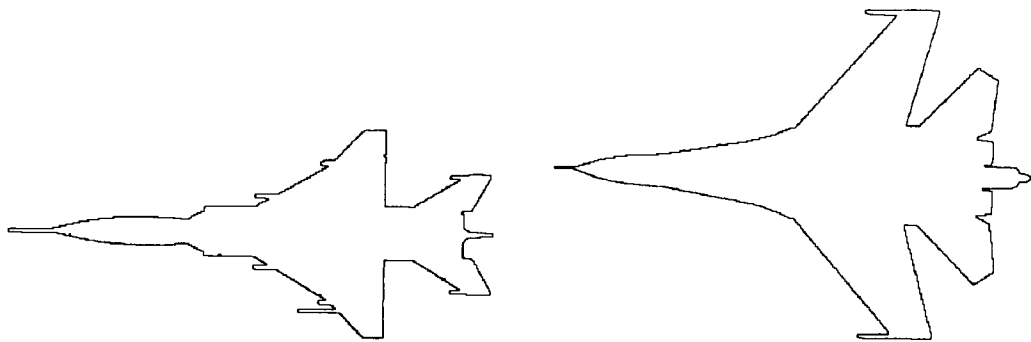
### Fighter Aircraft Sample Contours











## **Vita**

Joel Wesley Reed was born in Knoxville, Tennessee, on June 11, 1968. He attended grade schools in Knoxville and graduated from Farragut High School in June 1986. In August 1986, he entered the Liberal Arts College at The University of Tennessee, Knoxville. In May 1993, he received the Bachelor of Science degree with a major in Computer Science.

In August 1993, he entered the Graduate School at The University of Tennessee, Knoxville. He completed the requirements for the Master of Science degree with a major in Computer Science, in December 1995.